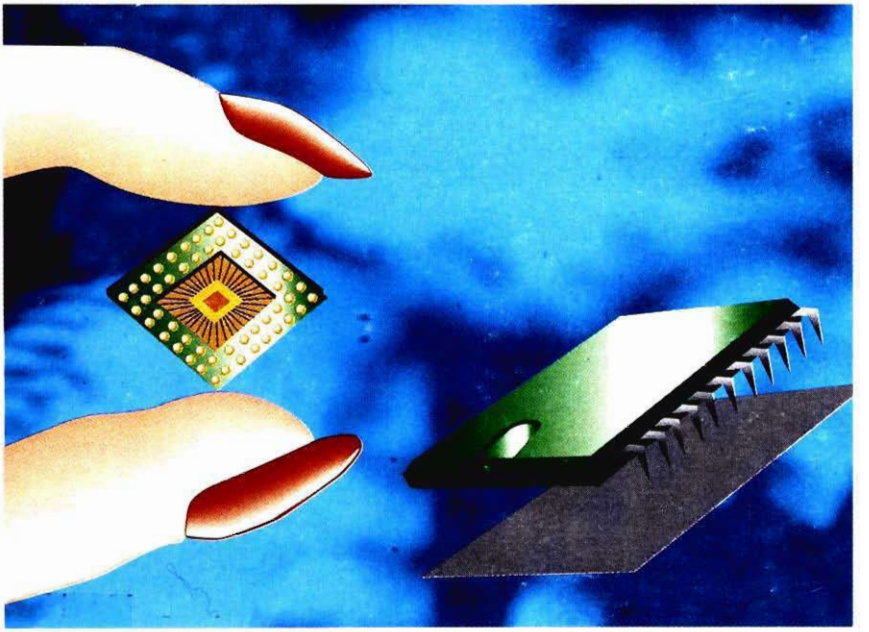
Prácticas de laboratorio de sistemas digitales



José Vega Luna • Gerardo Salgado Guzmán
Roberto Sánchez González

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

José Ignacio Vega Luna

Ingeniero electrónico y maestro en ciencias de la computación en la UAM-Azcapotzalco. En la misma institución ha impartido materias de circuitos lógicos, sistemas digitales y organización de máquinas digitales.
En la Universidad Nacional de Ingeniería de Managua, Nicaragua, impartió cursos de microprocesadores. Actualmente sus áreas de trabajo son: redes de computadoras, sistemas operativos y aplicaciones de microprocesadores y microcontroladores.

Gerardo Salgado Guzmán

Realizó estudios de ingeniería electrónica en la UAM-Azcapotzalco. En ella ha impartido materias de circuitos lógicos, sistemas digitales, laboratorio de sistemas digitales y laboratorio de máquinas digitales. Actualmente sus áreas de trabajo son: redes novell y aplicaciones de microprocesadores y microcontroladores.

Prácticas de Laboratorio de Sistemas Digitales

COLECCIÓN
Libros de texto y Manuales de prácticas
SERIE
Material en apoyo a la docencia
(Teoría y prácticas de laboratorio; problemarios)

Prácticas de Laboratorio de Sistemas Digitales

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Roberto Sánchez González

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

Primera edición 1996

ISBN 970 620 739 2

Impreso en México

Índice

Agradecimientos

A los profesores Armando Jiménez Flores, Rossen Petrov y Raymundo Lira, por haber dedicado parte de su tiempo, de manera desinteresada, a la revisión de este juego de práctica.

Los autores
abril de 1994

Prólogo

El objetivo principal de esta serie de prácticas es brindar un apoyo al estudiante en el aprendizaje de *microprocesadores* de 8 bits en sus diferentes interfaces y circuitos de soporte necesarios para configurar un sistema digital, en particular con el *Microprocesador Z80*, lo cual corresponde a la *U.E.A. Laboratorio de sistemas digitales*. La parte teórica correspondiente al microprocesador y circuitos utilizados en estas prácticas debe ser cubierta en la U.E.A de *Sistemas digitales*, sin embargo, cada práctica tiene una introducción teórica que le da al estudiante los conceptos básicos y algunas ideas necesarias para la realización de la misma.

Se pretende también que al terminar de realizar las prácticas, se cuente con un sistema digital completo que contenga: interfaces para transmisión de datos en forma serie y paralelo, interrupciones y temporización, así como convertidores digital-analógico y analógico-digital, integrando uno de estos elementos funcionales en cada práctica. Utilizando en todas las prácticas la *lógica positiva*.

Finalmente, cabe mencionar que esta guía es el fruto del trabajo de varios años, en los cuales el planteamiento, realización y depuración de las prácticas se debe en gran medida a los profesores que han impartido la materia de *Laboratorio de sistemas digitales* y a los comentarios y sugerencias de los alumnos que la han cursado en los diferentes trimestres. Esta guía de ninguna manera pretende sustituir el curso teórico de *Sistemas digitales*.

Práctica 1

El microprocesador Z80

Objetivos

- Aprender a usar un programa ensamblador.
- Diseñar y construir un sistema mínimo digital con el Microprocesador Z80 y probarlo con un programa que tenga acceso a memoria.

Trabajo a desarrollar

Diseñar y construir un sistema digital configurado alrededor del *Microprocesador Z80*, el cual únicamente cuente con los siguientes bloques funcionales: *CPU, memoria EPROM, circuito de reloj y circuito de reset*.

El programa debe de estar ejecutando de manera continua (un 'loop') instrucciones de salida o entrada: *OUT* o *IN* [por ejemplo: *OUT(XXH,A)*]:

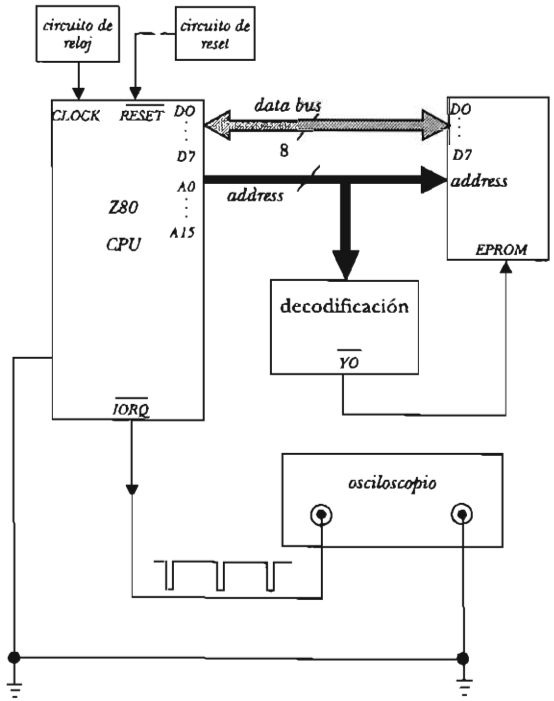
```
        ORG 000H
        NOP
        NOP
        NOP
LOOP:   OUT (00H), A
        JR LOOP
        END
```

El objetivo principal de este ejercicio es que se vea que el *Microprocesador Z80* esté realizando el acceso correcto a la memoria *EPROM*, la cual estará ejecutando un programa que indefinida y periódicamente active la salida *IORQ* de la *CPU*. Es decir, si el *Z80* está leyendo correctamente la *EPROM*, ejecutará el programa almacenado, lo cual se puede comprobar colocando una punta del osciloscopio en el pin *IORQ* del *Z80* y observando pulsos periódicos hacia cero.

Es en verdad un programa y un sistema sencillo, en el que existen muy pocos bloques funcionales que en el caso de no operar correctamente es muy fácil buscar y encontrar cuál de los bloques está fallando.

Este sistema es en realidad la base o plataforma de la cual surgen las siguientes prácticas.

Figura 1.1 Diagrama de bloques del sistema completo



Práctica 2

Ensamblador Z80-parte I

Objetivos

- Aprender las bases de la programación con lenguaje ensamblador Z80.
- Adquirir experiencia en el desarrollo de un sistema digital configurado alrededor de un microprocesador.

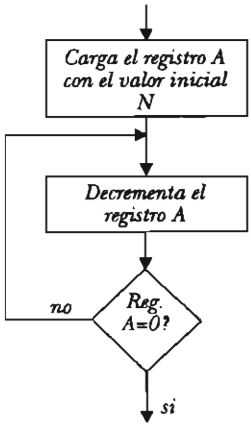
Introducción teórica

Rutinas de retardo o tiempos de espera

En muchos sistemas digitales es necesario generar retardos, lazos o 'loops' de tiempo para esperar que suceda un evento, durante este retardo generalmente el *microprocesador* permanece ocioso. Se pueden lograr estos retardos por software, usando las instrucciones del tipo *NOP* (No-operation) del procesador en conjunción con instrucciones de carga, decremento y salto.

Un retardo de tiempo por software se puede realizar con una subrutina que tenga unas cuantas instrucciones que realicen lo siguiente:

Figura 2.2 Diagrama de flujo para una rutina de retardo básica



Para calcular el tiempo que tarda en ejecutarse la subrutina y por lo tanto un retardo, deberán de tomarse muy en cuenta el tipo de instrucciones utilizadas. Cada instrucción dura o consume un tiempo de ejecución especificado en estados T o ciclos de reloj, los cuales incluyen tanto el ciclo fetch como el ciclo de ejecución de la instrucción.

Para diseñar una subrutina de retardo de un tiempo conocido lo único que debe calcularse, según el diagrama de flujo anterior, es el valor inicial que se le cargará al contador, el cual determina el número de iteraciones del lazo. Por ejemplo, si se utilizan las siguientes instrucciones para implementar la subrutina de retardo:

		Ciclos T
	LD A, N	(7)
LOOP:	DEC A	(4)
	JP NZ, LOOP	(7) si Z=1 (12) si Z=0

La duración del retardo es la suma del tiempo que consume la ejecución de la instrucción *LD A, N*; más el tiempo que consume la instrucción

DEC A multiplicado por *N* (ya que se ejecuta *N* veces); más el tiempo que consume la instrucción *JP NZ, LOOP* multiplicado por N-1 (el salto a la etiqueta LOOP se ejecuta N-1 veces); más el tiempo que consume la instrucción *JP NZ, LOOP* cuando no se ejecuta el salto (Z=1).

Para el caso de un *Microprocesador Z80* trabajando a 2 MHz, esta subrutina tardaría en ejecutarse *T=500ns[7+4N+(N-1)12+7]=500ns[16N+2]*.

Si el registro A se carga con 10, la subrutina tardaría en ejecutarse *8 microsegundos*.

Debe notarse que si el valor inicial del contador es 00H, el número de iteraciones del lazo será de 256, ya que cuando se decrementa por primera vez el contador, su valor pasa de 00H a FFH.

Por otra parte, cuando se requiere una precisión de tiempo que no se puede lograr sólo utilizando las instrucciones anteriores, es muy común agregarle a la subrutina instrucciones del tipo *NOP* para obtener el valor deseado, ya que las instrucciones *NOP* no afectan banderas, y además su código de operación es pequeño (no consumen mucha memoria) y mantienen realmente ocioso al *microprocesador*.

Así también, si se requieren tiempos más grandes no es muy práctico agregarle solamente instrucciones *NOP* a la subrutina, sino que más bien se puede realizar una subrutina que use un contador de iteraciones de más de 8 bits (por ejemplo un registro par del *Z80*):

Ciclos T

```
        LD BC,N       (10)   ;Inicializa al contador.
LOOP:   DEC BC         (6)   ;Decrementa al contador.
        LD A, B        (4)   ;Checa si el valor del contador es
        OR C           (4)   ;cero (B=C=00H).
        JP NZ, LOOP    (7)   ;Tarda 7 ciclos T si Z=1 o 12 ciclos T
                               si (7) Z=0.
```

en donde ahora, por supuesto, deberá de tomarse en cuenta el tiempo que consumen las instrucciones adicionales de la subrutina.

Otra manera de generar retardos mayores es con lazos anidados, uno dentro de otro:

```
        LD B, M
        LD A, N
LAZO A: DEC A
        JP NZ, LAZO A
        DEC B
        JP NZ, LAZO B
```

En donde el lazo del *registro A* se ejecuta *N* veces y el lazo del *registro B* se ejecuta *M* veces y la subrutina tiene una duración de (*M*•*N*) veces.

Finalmente, podemos decir que los retardos de tiempo no son una manera muy elegante de que el *microprocesador* espere por un evento manteniéndolo ocioso, ya que esto puede lograrse también manteniéndolo activo realizando otras operaciones e interrumpirlo cuando suceda el evento esperado usando los mecanismos de interrupción disponibles. Así también, si se desea que el *microprocesador* esté monitoreando periódicamente un proceso, se puede utilizar un circuito temporizador externo (como se observará en una práctica posterior en la que se usa un *Z80-CTC*) que le indique al *microprocesador* que cada cierto tiempo debe checar el proceso, lo cual es en realidad una solución por hardware.

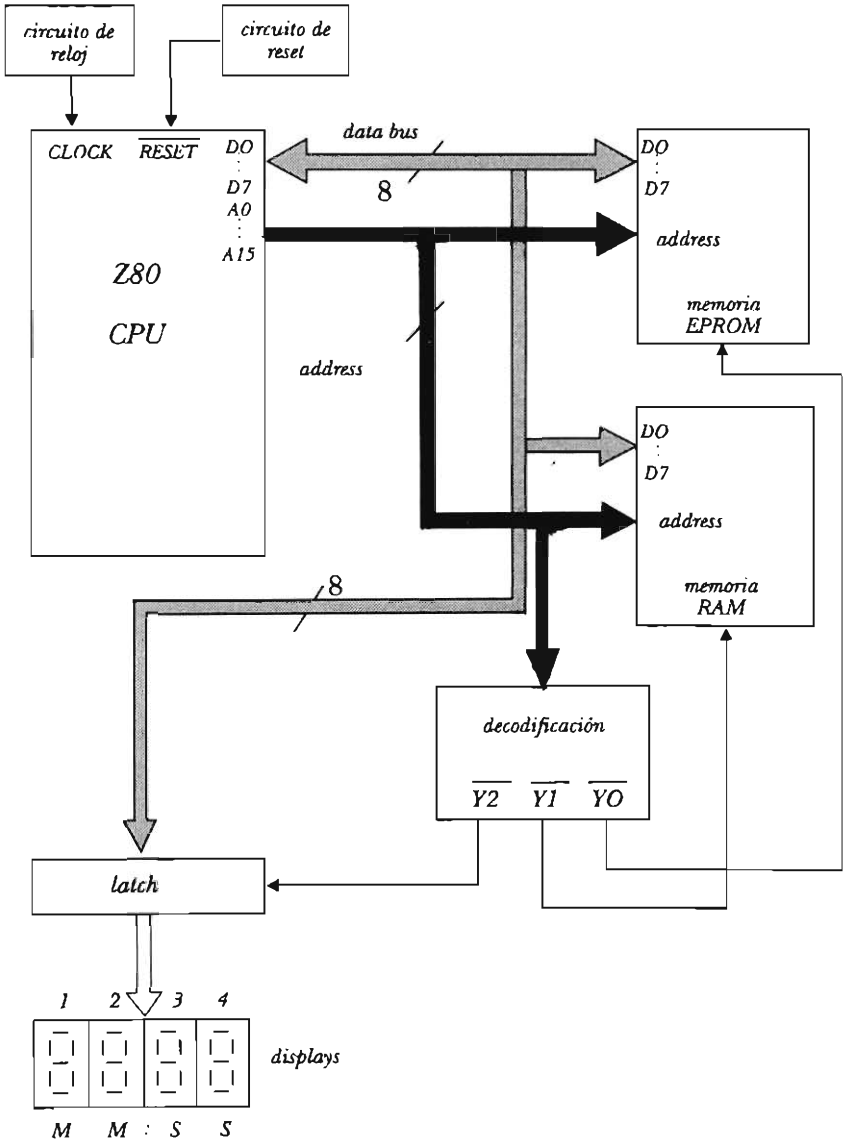
Trabajo a desarrollar

Diseñar y construir un sistema digital configurado alrededor del *Microprocesador Z80*, el cual se encuentre mostrando un reloj de tiempo real en cuatro *displays* de siete segmentos.

No se deberán utilizar interrupciones y el formato de la hora será el siguiente:

MM: SS MM: minutos
SS: segundos

Figura 2.2 Diagrama de bloques del sistema completo



Práctica 3

Ensamblador Z80-parte II y transmisión paralelo

Objetivos

- Conocer cómo se lleva a cabo en un sistema digital, configurado alrededor del Microprocesador Z80, un mapeo de puertos de E/S.
- Aprender a programar y utilizar de manera práctica una interfaz paralela.

Introducción teórica

Comunicación de datos en paralelo

- La comunicación de datos en paralelo se realiza transfiriendo todos los bits del dato al mismo tiempo y por líneas separadas. Su principal ventaja con respecto a la transmisión serie, es que se pueden lograr velocidades de transmisión muy altas debido al uso de varias líneas. La desventaja es, por supuesto, el costo extra de las líneas, ya que el costo se incrementa con la distancia. La comunicación en paralelo se usa para cortas distancias y altas velocidades de transmisión.
- El estandard para la comunicación en paralelo es la interface centronics. La transmisión en paralelo se puede realizar simplemente

escribiendo el dato en el *registro de datos* de una interfaz o leyendo un dato de entrada del mismo registro de la interfaz, o también puede realizarse usando las señales de control y flujo de información o de *handshaking*. Normalmente, un carácter se transfiere a un instante de tiempo y no existe el problema de determinar el final del carácter o buscar el inicio de la transmisión. No hay un formato bien definido como en la transmisión serie, pero si existe temporización (un ordenamiento en el tiempo) de las señales que coordinan la actividad entre un periférico y una interfaz, la transmisión paralelo se debe de considerar síncrona. Si sólo se utilizan señales de control de flujo de la información, la transmisión debe ser considerada asíncrona.

- Una interfaz paralela puede diseñarse sólo para operaciones de salida, de entrada o ambas con un conjunto de líneas separadas, o realizando ambas operaciones con un mismo conjunto de líneas bidireccionales. Por ejemplo, una interfaz se puede usar para enviar datos a una impresora o para recibir datos de una lectora de cinta de papel.

En esta práctica en particular, usaremos una interfaz paralelo programable de *Intel* de tres puertos de ocho líneas cada uno, comúnmente conocida como *interfaz para periférico programable 8255A (PPI: Programmable peripheral interface)*. Durante la explicación que haremos a continuación del *8255A* es recomendable que el lector se apoye en las hojas de datos del manual del fabricante.

La interfaz para periférico programable

El *8255A* es un dispositivo de entrada/salida programable de propósito general diseñada para ser utilizada con *microprocesadores de Intel*, pero puede usarse también con cualquier otro *microprocesador* o *microcontrolador* de otro fabricante sin mayores problemas. Este circuito integrado tiene 24 líneas que pueden programarse de manera

individual en dos grupos de 12 y usarse en tres diferentes modos de operación. En el primer modo (*el modo 0*), cada grupo de 12 líneas se puede programar en conjuntos de cuatro líneas que puede ser de entrada o de salida. En el *modo 1*, cada grupo se puede programar para tener ocho líneas de entrada o de salida y de las cuatro líneas restantes, tres se usan como señales de control de interrupción y de transferencia de datos o de *handshaking*. El tercer modo de operación (*el modo 2*) permite al *8255A* configurar ocho de sus líneas como un bus bidireccional y a otras cinco líneas, tomadas del otro grupo, como señales de control de flujo de datos (*handshaking*).

Descripción del 8255A

El *8255A* tiene tres puertos de ocho líneas cada uno (*puerto A*, *puerto B* y *puerto C*) y un llamado *registro de control*. Los tres puertos sirven para transferir datos en forma paralela entre un periférico y un *microprocesador*. El *registro de control* sirve para recibir del software del sistema una palabra de control que programa el modo de operación del *8255A*. Así pues, el *8255A* utiliza cuatro puertos de entrada/salida del sistema.

El *8255A* divide de manera lógica las 24 líneas de sus tres puertos en dos grupos funcionales:

- *El grupo A*: compuesto por el *puerto A* (PA0-PA7) y las cuatro líneas más significativas del *puerto C (PC4-PC7)*.
- *El grupo B*: compuesto por el *puerto B* (PB0-PB7) y las cuatro líneas menos significativas del *puerto C (PC0-PC3)*.

Operación y selección de modo del 8255A

Tiene tres modos de operación básicos que se pueden seleccionar por software:

Modo 0	Entrada/salida básica.
Modo 1	Entrada/salida con *handshaking*.
Modo 2	Bus bidireccional.

El usuario puede realizar una gran variedad de combinaciones de modos de operación de los puertos utilizados para obtener la estructura de entrada/salida deseada. Por ejemplo, el *grupo A* se puede programar en *modo 1* para monitorear un teclado o un lector de cinta por interrupción, mientras que el *grupo B* se puede programar para monitorear un grupo de interruptores (on/off) o desplegar resultados de sistema bajo control o supervisión.

Modos de operación del 8255A

El modo 0 (entrada/salida básica). Por medio de este modo se pueden realizar operaciones de entrada y de salida simples con cada uno de los tres puertos. El *8255A* no proporciona en este modo líneas de control de la transferencia de la información (*handshaking*). El dato simplemente es escrito o leído en el puerto. Los puertos usados como salida en este modo tienen un latch en cada una de sus líneas y los puertos de entrada no lo tienen.

El modo 1 (entrada/salida con handshaking). En este modo el *8255A* proporciona dos puertos (el *puerto A* y el *puerto B*) para entrada/salida con *handshaking*. Estos dos puertos utilizan algunas líneas del *puerto C* como señales de *handshaking*.

Cuando el *puerto A* o el *puerto B* se programan en *modo 1* y de entrada, manejan las siguientes tres líneas de control o de *handshaking*:

$\overline{STB}$ (Entrada de muestreo o de strobe). Entrada, activo bajo. Cuando tiene un '0' lógico, el dato que se encuentra presente en las líneas del puerto se transfiere a un latch de entrada interno.

IBF (Buffer de entrada lleno-Input buffer full). Salida, activo alto, que indica que el dato de entrada ha sido tomado de las líneas del

puerto y cargado en el latch interno, es un reconocimiento o respuesta de la señal de *STB*. *IBF* es activada por la señal *STB* y desactivada por la entrada *RD* (lectura-*read*) del *8255A* es decir, cuando la *CPU* lee el dato de entrada del latch del *8255A*.

INTR (Solicitud de interrupción-Interrupt request). Salida, activo alto, que puede ser usada por un dispositivo externo para interrumpir a la *CPU* y solicitar su servicio.

Por otra parte, los *puertos A* y *B* tienen un flip-flop interno llamado *INTE A* e *INTE B*, respectivamente, que permite habilitar o deshabilitar las solicitudes de interrupción hacia la *CPU*. El usuario puede tener acceso a este flip-flop utilizando una característica del *8255A* llamada *bit set/reset* que analizaremos más adelante.

La salida *INTR* se activará cuando las señales $\overline{STB}$ e *IBF* sean '1' lógico y además el flip-flop *INTE* tenga también '1' lógico. La salida *INTR* se desactiva con la caída de $\overline{RD}$.

Cuando el *puerto A* o el *puerto B* se programan en *modo 1* como salida, manejan las siguientes tres líneas de control de flujo de datos o de *handshaking*:

$\overline{OBF}$ *(Bufer de salida lleno-Output buffer full)*. Salida, activo bajo, Un '0' lógico en esta entrada le indica al *8255A* que el dato de salida en el puerto del *8255A*, se desactiva cuando la señal de entrada de $\overline{ACK}$ pasa a '0' lógico.

$\overline{ACK}$ *(Entrada de reconocimiento-Acknowledge)*. Entrada, activo bajo. Un '0' lógico en esta entrada le indica al *8255A* que el dato de salida ya ha sido aceptado o tomado del *puerto A* o *B* por el periférico.

INTR (Solicitud de interrupción-Interrupt request). Salida, activo alto. Esta señal se puede usar para solicitar interrupción a la *CPU* por parte de un periférico cuando este último ha tomado el dato transmitido por la *CPU*. Esta señal se activa cuando la entrada $\overline{ACK}$ es '1' lógico, la salida $\overline{OBF}$ es '1' lógico y el flip-flop *INTE* del puerto es '1' lógico.

El modo 2 (bus bidireccional). Tal vez es el más poderoso del *8255A* y es apropiado para la comunicación con un dispositivo periférico

por medio de un bus bidireccional de ocho bits. Cuenta con señales de control de flujo de la información e interrupción, al igual que en el *modo 1*. El *Grupo A* programado en este modo usa cinco líneas del *puerto C* para implementar estas señales de control de flujo de la información, por lo que el *grupo B* (el *puerto B* en específico) no se puede programar en *modo 2* y sólo resta programarlo en *modo* 0 o en *modo 1*. Las cinco señales de control que usa este bus bidireccional son las siguientes:

Operaciones de salida:

Para operaciones de salida, se utilizan las señales de control para el flujo de información que se usa en el *modo 1: $\overline{OBF}$ y $\overline{ACK}$.*

Operaciones de entrada:

En este caso, también se usan las señales de control para el flujo de información que se utiliza en el *modo 1* (entrada): *$\overline{STB}$ e IBF.*

INTR (Solicitud de interrupción-Interrupt request). Salida, activo alto. Un '1' lógico en esta línea se puede usar para solicitar interrupción a la *CPU* para operaciones de entrada o de salida. Ya que esta señal se activa para dos tipos de operaciones, el usuario tiene la opción de habilitar o deshabilitar las interrupciones individualmente, tanto para una operación de entrada como para una de salida, es decir, la interrupción puede ser generada tanto por una operación de entrada como por una de salida o ambas.

Una diferencia muy importe del *modo 2* con respecto al *modo 1* es que una operación de salida, el dato escrito por la *CPU* en el *puerto A*, no aparece en las líneas del *puerto A* hasta que el periférico no envíe o active la señal de *$\overline{ACK}$ (Acknowledge)*, asegurándose así que el mismo periférico ya está listo para recibir un dato. La secuencia de activación de las señales para operaciones de entrada en el *modo 2* funcionalmente es igual que en el *modo 1.*

La característica de habilitar o deshabilitar un bit

También se le conoce como la operación de *bit set/reset* y sirve para escribir un '1' o un '0' lógico a una línea específica del *puerto C*, y es muy útil cuando las líneas de este puerto se usan como indicadoras de estado o control del *puerto A* o *B*.

Para usar esta característica del *8255A*, la *CPU* deberá escribir una palabra llamada de *bit set/reset* en el *registro de control*. Esta palabra se diferencia de una de *control* porque la primera debe tener un '0' lógico en el bit más significativo.

Así también, la palabra de *bit set/reset* contiene la información que indica a cuál de las 8 líneas del *puerto C* se desea tener acceso y el valor que se desea escribir en la línea seleccionada.

Por otra parte, hemos dicho que para habilitar o deshabilitar las interrupciones existe un flip-flop interno asociado a los *puertos A* y *B*. Estos flip-flop's se llaman *INTE A* para el *puerto A* e *INTE B* para el *puerto B*, y están asociados a la línea del *puerto C*, usada también para la señal de $\overline{STB}$ en las operaciones de entrada y a la línea usada para la señal de $\overline{ACK}$ para operaciones de salida. Es decir, no se puede escribir realmente sobre las líneas de $\overline{STB}$ y $\overline{ACK}$ (que son entradas) sino que se escribe sobre el flip-flop de interrupciones asociado a la línea del *puerto C*. Las líneas del *puerto C* asociadas a estos flip-flop's son las siguientes:

Modo 1 (entrada):
- Flip-Flop INTE A : PC4
- Flip-Flop INTE B : PC2

Modo 1 (salida):
- Flip-Flop INTE A : PC6
- Flip-Flop INTE B : PC2

Modo 2:
- Flip-Flop INTE 1 (operaciones de salida) : PC6
- Flip-Flop INTE 2 (operaciones de entrada) : PC4

Entonces, podemos decir finalmente que el *8255A* es una herramienta hardware muy poderosa para comunicar un periférico paralelo con un sistema digital de forma paralelo sin necesidad de utilizar lógica muy compleja.

Trabajo a desarrollar

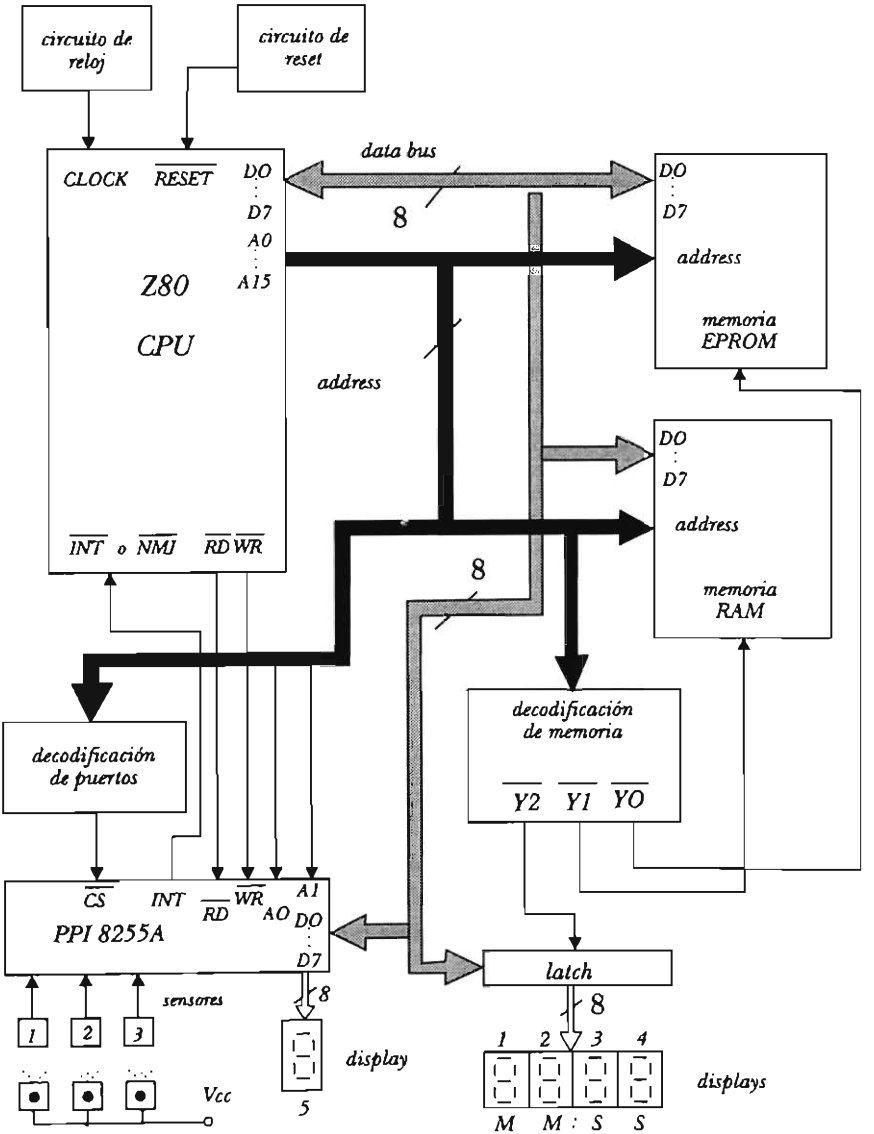
Diseñar e implementar un sistema digital mediante el cual se pueda interfazar tres sensores (fotodiodos, fototransistores, etcétera) al *Microprocesador Z80*, de forma tal que cuando algún objeto cruce por alguno de ellos, se deberá de solicitar interrupción a la *CPU*. La rutina de atención a la interrupción mostrará en un *display* de siete segmentos (el *display 5*) el número del sensor por el cual cruzó el objeto. En caso de que dicho objeto haya cruzado por más de un sensor, el *display 5* mostrará de manera alternada cada número durante un segundo.

El programa principal deberá estar actualizando un reloj de tiempo real que se mostrará en cuatro *displays* de siete segmentos (ya hecho en la práctica anterior).

Se usará un *PPI 8255A* para interfazar los sensores y el *display 5* adicional con el sistema utilizado en la práctica anterior.

Pueden utilizar cualquier *modo de interrupción del Z80*, así como cualquier tipo de sensor (fotodiodo, fototransistor) y deberán de mapear el *PPI* en el espacio de direccionamiento de *puertos de E/S*.

Figura 3.1 Diagrama de bloques del sistema completo



Práctica 4

Transmisión serie

Objetivos

- Conocer las bases y aspectos prácticos de la transmisión de datos serie.
- Aprender a programar una interfaz serie.

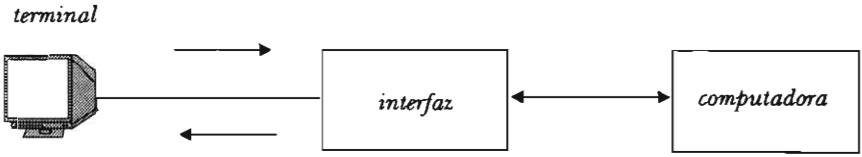
Introducción teórica

Comunicación de datos en serie

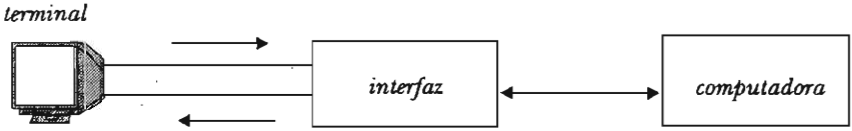
Muchos dispositivos periféricos transfieren información desde o hacia una computadora en forma serie, es decir, bit a bit por medio de un par de conductores o un canal de comunicaciones. Cada bit ocupa un intervalo de tiempo teniendo una longitud específica. Existe una gran variedad de interfaces serie, las cuales típicamente tienen un *registro de estado* que contiene la información sobre los errores y el estado de la transmisión, y un *registro de control* que contiene la información que determina el modo de operación de la misma. Generalmente, estos dispositivos tienen líneas separadas tanto para transmitir como para recibir la información en forma serie. La interfaz tiene líneas diferentes: cuando el sistema de comunicación puede transmitir y recibir al mismo tiempo en líneas diferentes se le llama *full-duplex*

y si utiliza la misma línea para transmisión y recepción se le llama *half-duplex*.

Figura 4.1 Transmisión half-duplex y full-duplex



half-duplex



full-duplex

Hay básicamente dos tipos de comunicación serie: *asíncrona y síncrona*. En la comunicación *asíncrona* existen patrones de bits especiales que separan a los caracteres a transmitir. En la comunicación *síncrona* los caracteres son enviados uno tras otro sin bit de paridad y de parada, es decir, el DTE y DCE mandarán cada uno su reloj de sincronía a través de las líneas TxCL y RxCL de la RS232. En la comunicación *asíncrona* con protocolo de comunicación síncrono se utiliza los caracteres de sincronía *sync* al inicio de cada paquete enviado.

La interfaz serie 8251A

Podemos decir que entre las interfaces serie más comúnmente usadas se encuentran el *8251A* de *Intel*, el *Z80-SIO*, el *Z80-DART* de *Zilog* y el *ACE 8250* de *Western Digital*, por mencionar algunos. En esta práctica usaremos el *8251A* de *Intel*, también conocido como *USART (Receptor/transmisor universal síncrono/asíncrono)*, para la transmisión de datos serie asíncrona.

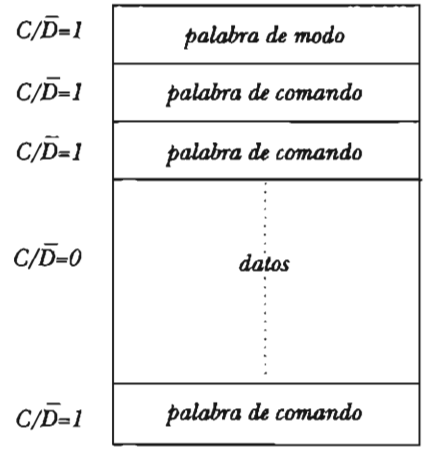
El *8251A* convierte los datos recibidos de una *CPU* en forma paralelo a un formato de bits en forma serie en la transmisión y también convierte una cadena de bits que se reciben en forma serie a un dato de 8 bits listo para ser leído por un *microprocesador*. Es decir, el 8251A tiene tanto un transmisor como un receptor que pueden operar de manera independiente al mismo tiempo o *full-duplex*. Durante la explicación que haremos a continuación del *8251A* es recomendable que el lector se apoye en las hojas de datos del manual del fabricante.

Operación y selección de modo del 8251A

Para programar las funciones de este circuito se utilizan básicamente dos palabras: una *palabra de modo* y una *palabra de comando*. Ambas palabras son necesarias para definir las características del tipo de comunicación serie que realizará el *8251A*, pero existe una secuencia que debe seguirse al enviar la *CPU* estas palabras hacia el *8251A*, la secuencia a seguir es la siguiente de la figura 4.2.

Para comenzar la transmisión o recepción de datos serie, se le debe enviar el *8251A* una *palabra de modo* inmediatamente después de la operación de inicialización (*reset*) ya sea por software o por hardware y a continuación una *palabra de comando*. Todas las palabras de control que la *CPU* envíe el 8251A a continuación de la *palabra de modo* se interpretarán como *palabras de comando* y la forma de salirse de este lazo es aplicándole un *reset* al *8251A*.

Figura 4.2 Secuencia de programación del 8251A



El *8251A* tiene un *registro de control*, al cual se le debe enviar o escribir (la entrada C/ $\overline{D}$ = '1') la *palabra de modo* y la *palabra de comando*, y un *registro de datos*.

Cuando la *CPU* lee este *registro de control* en realidad lo que hace es leer el estado del *8251A* y saber si han ocurrido errores en la comunicación serie que requieran la atención del procesador. La información de estado que se obtiene en esta lectura la veremos más adelante.

Cuando la *CPU* escribe al *registro de datos* del *8251A* (la entrada C/$\overline{D}$ = '0') le estará enviando al *buffer de transmisión* un dato que desea transmitir en forma serie y cuando lee este *registro* estará leyendo del *buffer de recepción* un dato que previamente arribó en forma serie.

Modos de operación del 8251A

El *8251A* puede transmitir y recibir datos de manera *asíncrona* o *síncrona*, de tal forma que puede ser visto como un dispositivo con

dos componentes separados compartiendo el mismo encapsulado, uno *asíncrono* y el otro *síncrono*. El modo de operación puede cambiarse solamente después de aplicarle un *reset* al circuito integrado.

En esta práctica se utilizará la transmisión de datos serie en forma asíncrona en la que el *8251A* envía al inicio de cada carácter un *bit de inicio* o *bit de start* ('0' lógico), seguido de los bits del mismo dato, a continuación un *bit de paridad* (si el usuario lo programa), que indica si la paridad del dato transmitido es par o impar, y finalmente el o los *bits de parada* o *bits de stop*. Todos estos bits componen una cadena que es enviada por la salida *TxD* del *8251A* bit a bit y a una razón de 1, 1/6 o 1/64 (según sea programada) de la velocidad de la señal suministrada por la entrada *TxC*. Cuando la *CPU* no ha enviado datos a transmitir al *8251A*, su salida *TxD* permanece en un nivel activo alto, también llamado un estado de *marca*, a menos que el usuario programe que se envíen *caracteres de ruptura* o *caracteres break*.

Todas estas características, así como el número de *bits de parada*, se programan en la *palabra de modo* del *8251A*.

En la recepción de datos asincrona, los bits entran el *8251A* por medio de la línea *RxD*, la cual normalmente se encuentra en nivel activo alto. Una transición de bajada en esta línea le indica al *8251A* que ha llegado un *bit de inicio*, por lo que a continuación comienza a muestrear la línea *RxD* y recibir los bits del dato, el *bit de paridad* (si existe) y el *bit de parada*, siempre en la mitad de su valor nominal. Si el *bit de paridad* no corresponde al programado, se activa una bandera del *8251A* llamada bandera de *paridad* (*PE: Parity error flag*). Si se detecta un '0' lógico en el *bit de parada*, se activa la bandera de *error de formato* (*FE: Framing error flag*).

Cuando el *8251A* ha recibido un dato en forma serie, lo transfiere al *buffer de recepción*. Si a continuación, se termina de recibir otro dato en forma serie y la *CPU* no ha leído el que se encuentra en el *buffer de recepción*, se activa una *bandera de error de atención* (*OE: Overrun error flag*), y el dato que arriba se almacena en el buffer, perdiendo el carácter previo. Todas las banderas de error se pueden limpiar con un bit destinado para ello en la *palabra de comando*.

Así pues, podemos decir que la *palabra de modo* es la que define las características generales de operación del *8251A* y la *palabra de comando* controla la operación del modo seleccionado en la primera, realizando funciones tales como: habilitar/deshabilitar el transmisor y/o el receptor; limpiar las *banderas de error* y manejo de las señales de control de *MODEM*.

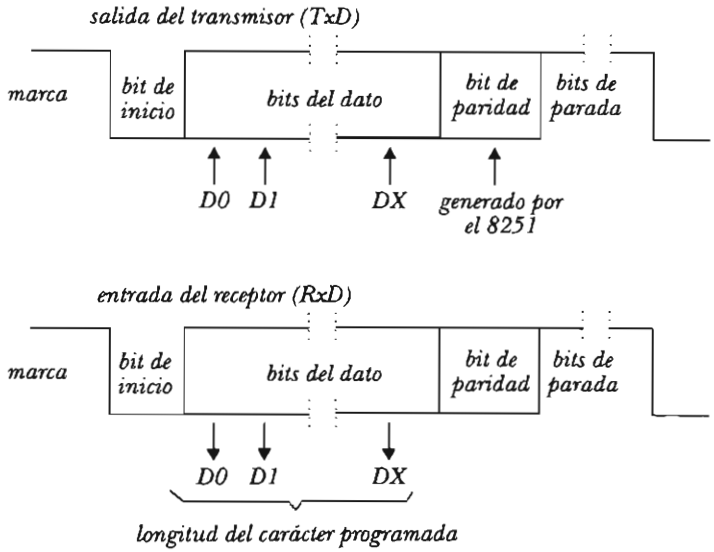
Lectura del estado (status) del 8251A

El *8251A* tiene la facilidad de que el programador puede leer en cualquier instante de tiempo el estado del dispositivo, con una operación de lectura cuando el pin $C/\overline{D}$ = '1', es decir, el *registro de control* del *8251A* no puede ser leído. Tres bits de la *palabra de estado* son las tres banderas de error que mencionamos anteriormente, y algunos de los otros bits restantes tienen un significado idéntico a su correspondiente pin de salida del *8251A* cuyo objetivo es que su estado pueda probarse también por software. (Figura 4.3.)

Trabajo a desarrollar

Adicionar al sistema digital de la práctica anterior una *interfaz serie 8251A (USART)* y realizar un programa que se encuentre desplegando un reloj de tiempo real y mostrando en un *display* (*display 5*) el número de sensor(es) activado(s) (práctica 3). Cada vez que se active un sensor se deberá de solicitar una interrupción a la *CPU Z80*, de tal forma que ésta deberá de transmitir hacia un bucle de retorno en forma serie (por medio del *8251A*) el código *ASCII* del número de sensor activado. Si al cabo de un tiempo (*time-out*) *T=10 segundos*, no se recibe el mismo código *ASCII* enviado anteriormente, se deberá de mostrar en el *display 5*, asociado a los sensores, un mensaje de *ERROR* (puede ser solamente una '*e*'), y esperará el sistema a que otro sensor sea activado.

Figura 4.3 La transmisión serie asíncrona



En el caso de que se active más de un sensor, se deberán de enviar los correspondientes códigos *ASCII* de los números de sensores y esperar a que dichos códigos sean recibidos (por la línea *RxD* del *8251A*) correctamente. Si alguno de estos códigos no se recibe directamente o si el *time-out* expiró sin haberse recibido dato alguno, se deberá de desplegar el mensaje de *ERROR* de todos modos.

Ejemplo: Si se activaron los sensores #1 y #3, pueden suceder los siguientes eventos:

Figura 4.4 No se recibe respuesta del primer código ASCII

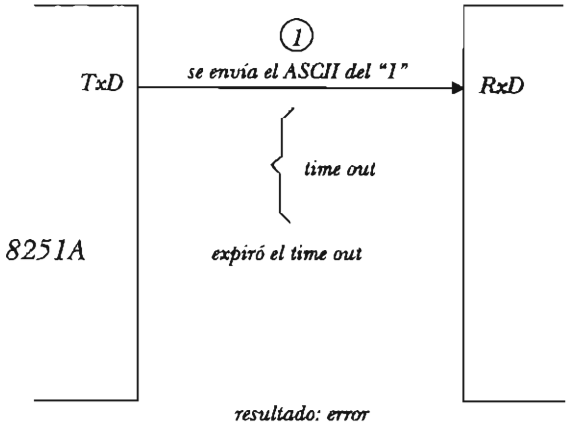


Figura 4.5 No se recibe respuesta del segundo código ASCII

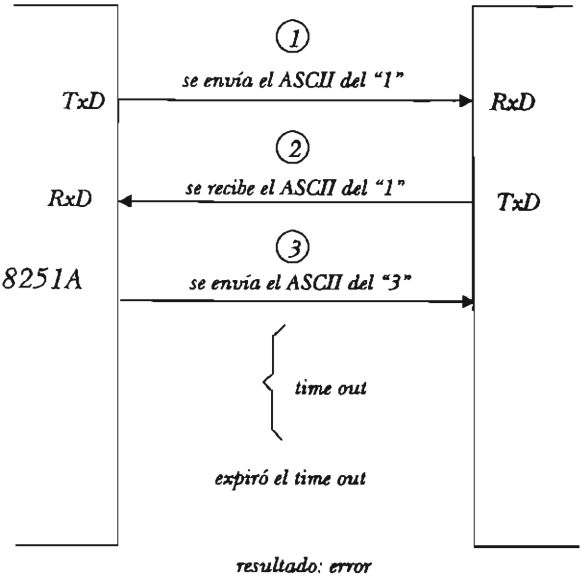
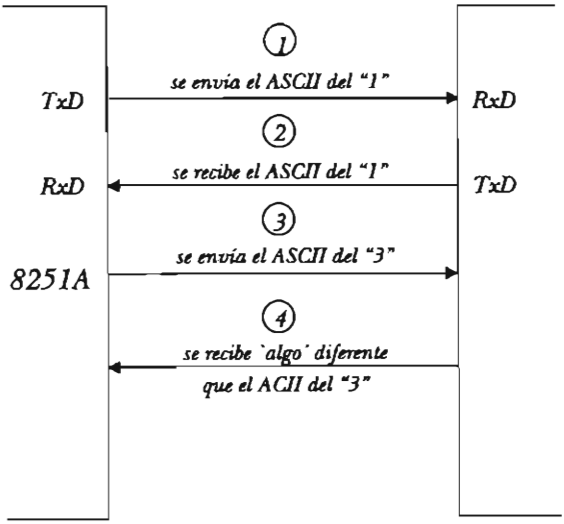
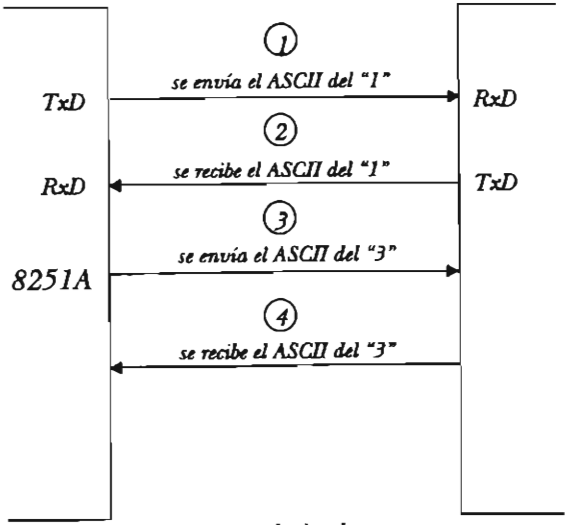


Figura 4.6 Se recibe un código ASCII diferente al enviado



resultado: error

Figura 4.7 Transferencia exitosa



resultado: ok

Figura 4.8 Diagrama de bloques del sistema completo

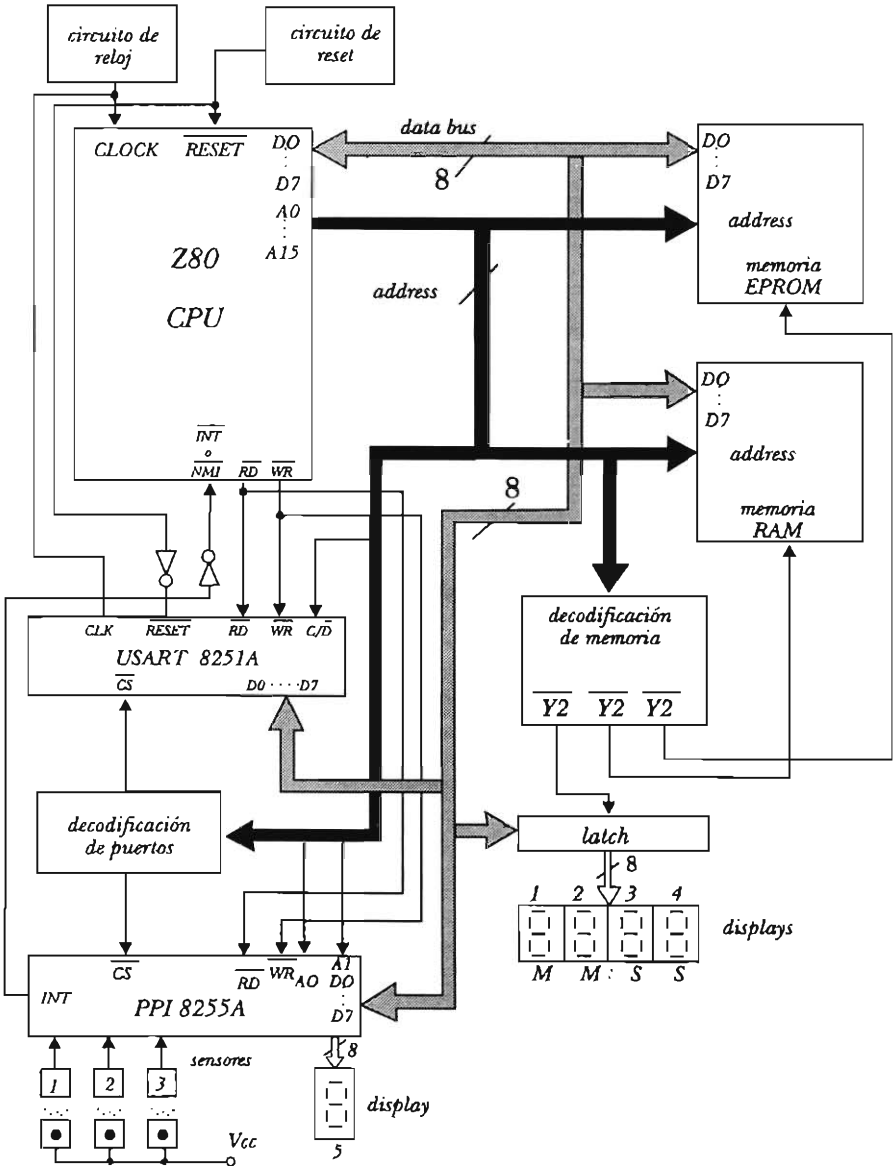
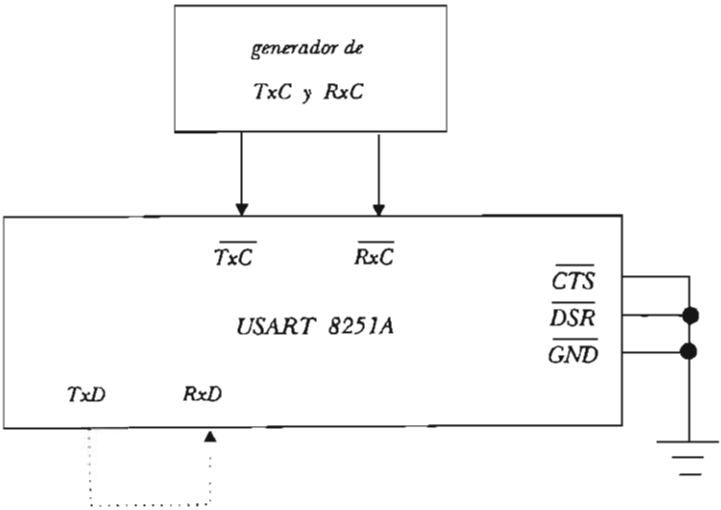


Figura 4.9 Simulación de la recepción de datos serie



Notas:

- Se puede colocar un puente entre las líneas TxD y RxD del USART 8251A para simular la recepción de datos serie.
- La velocidad de transmisión/recepción serie será de 1200 b.p.s.
- El 8251A trabajará en modo *full duplex*.

Práctica 5

Temporización (el Z80-CTC parte I)

Objetivos

- Aprender a programar y explotar los recursos ofrecidos por un circuito *counter/timer*.
- Adquirir experiencia en el manejo de circuitos temporizadores y contadores de eventos (*counter/timer's*).
- Manejar de manera correcta el modo 2 de interrupción de la Z80-CPU.

Introducción teórica

Temporizadores y contadores de eventos programables

Muy a menudo se necesitan dispositivos que marquen intervalos de tiempo tanto a un microprocesador como a dispositivos externos de un sistema digital, así también, que cuenten eventos externos y proporcionen la cuenta al *microprocesador*. Tales dispositivos se llaman *temporizadores/contadores de eventos programables (progammable interval timer/counters)* y entre sus aplicaciones podemos mencionar.

1. Interrumpir a un sistema operativo multiusuario para lograr la conmutación entre programas.

2. Enviar señales temporizadas con periodos programables a un dispositivo de entrada/salida (por ejemplo, un *convertidor digital/analógico*).
3. Generador de *baud-rate*.
4. Medir retardos de tiempo entre eventos externos.
5. Interrumpir al procesador después de que ha ocurrido una cantidad de eventos externos.

Un diagrama de bloques de un temporizador/contador de eventos típico se muestra en la figura 5.1.

El contador comienza a operar con un valor inicial y cuenta de forma descendente hasta llegar a 0. La entrada de *CLK* determina la rapidez del conteo, la entrada de *GATE* permite habilitar/deshabilitar la entrada de *CLK* y la salida *OUT* puede conectarse a una línea de solicitud de interrupción de un *microprocesador*. Cada vez que sucede un pulso activo en la entrada de *CLK* el contador se decrementa.

El modo de programación indica generalmente qué es lo que pasa cuando el valor del contador llega a 0.

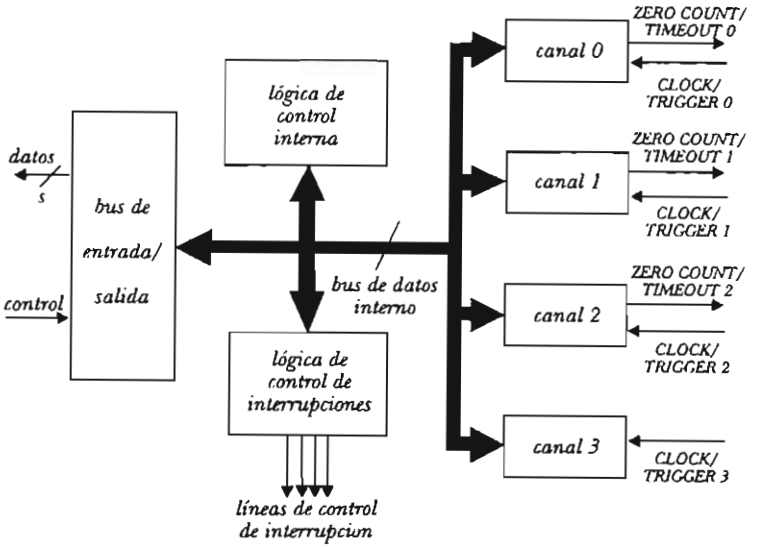
Ejemplo: Supongamos una aplicación de un temporizador usado en un sistema operativo de tiempo compartido, donde el reloj del sistema se conecta a la entrada de *CLK* y la salida *OUT* a una línea de solicitud de interrupción no-mascarable. El valor del *registro contador inicial* debe ser:

$$cuenta\ inicial = frecuencia\ de\ reloj \times T$$

donde T es el tiempo en segundos que el sistema operativo atiende a cada usuario, y el modo de operación debe ser tal que cuando el contador alcance el valor de 0, el contenido del *registro contador inicial* sea transferido al contador y se active la salida *OUT*.

Existen diferentes circuitos integrados que proporcionan en un mismo dispositivo una cantidad pequeña (de 2 a 4) de *temporizadores* y *contadores*, tales como el *Z80-CTC* de *Zilog* y el *8253* de *Intel*. En esta práctica utilizaremos el circuito *Z80-CTC* debido a que es el más

Figura 5.1 Diagrama de bloques de un temporizador/contador



apropiado para usarse con el *Microprocesador Z80* por ser de la misma familia, pero esto no implica que un *8253* no pueda conectarse a un *Microprocesador Z80*.

Durante la siguiente explicación del *Z80-CTC* es recomendable que el lector se apoye en las hojas de datos del manual del fabricante.

El circuito temporizador/contador de eventos Z80-CTC

El *Z80-CTC (Counter/timer circuit)* es un circuito cuyo nombre se debe a que realiza dos funciones básicas: *contador* y *temporizador*. Tiene cuatro canales independientes, funcionalmente idénticos, llamados *CH0*, *CH1*, *CH2* y *CH3*.

Los tres primeros canales tienen una entrada llamada *CLOCK/TRIGGER (CLK/TRG)* y una salida llamada *ZERO COUNT/TIME OUT (ZC/TO)*. El *canal 3* solamente tiene la entrada de disparo (*trigger*) *CLK/TRG*

debido a la limitación originada por ser un circuito integrado de 24 pines.

Modos de operación del Z80-CTC

El modo contador

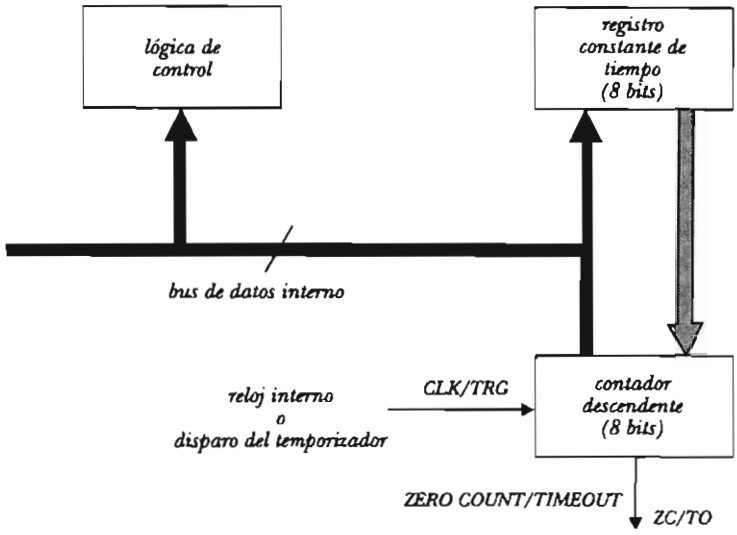
Cuando el canal del *Z80-CTC* se programa en modo contador se utiliza como contador de eventos externos o pulsos de reloj. Estos pulsos de reloj deben de entrar por la pata *CLK/TGR* del canal, cada vez que se detecte un pulso activo, se decrementa en uno el valor de una *constante de tiempo* que inicialmente fue programada en el *registro constante de tiempo* del canal. Esta constante es un número binario con un valor máximo de *FFH*. Cuando dicha *constante de tiempo* alcanza el valor de *00H*, se activa la salida *ZC/TO* del canal y el valor inicial de la *constante de tiempo* es recargada de manera automática en un *contador descendente* (*down counter*) del canal. El contador empieza a trabajar inmediatamente después de que el canal ha sido programado o inicializado.

Cuando se carga una nueva constante de tiempo en el *registro constante de tiempo* mientras el *contador descendente* está trabajando, la cuenta presente continuará hasta que alcance el valor de *00H*. Así también, el valor del *contador descendente* puede ser leído por el *microprocesador* en cualquier instante de tiempo para monitorear las operaciones del sistema. (Figura 5.2.)

El modo temporizador

Cuando un canal se programa en este modo de operación, cada 16 o 256 pulsos de la señal de reloj del sistema, presente en el pin *CLK* del *Z80-CTC*, se decrementa en uno la constante de tiempo originalmente programada. A este valor de 16 o 256 se le llama *prescalar* y divide

Figura 5-2 Diagrama de bloques de un contador típico



la frecuencia de la señal de reloj del sistema. Cuando el valor de la constante de tiempo llega al valor de *00H* suceden los mismos eventos que cuando el canal se programa en *modo contador*. En este caso el canal o temporizador comienza a operar cuando: se le aplica un pulso de disparo en su entrada *CLK/TRIG* o inmediatamente después de que ha sido cargada la *constante de tiempo*.

El periodo de tiempo del pulso de salida que proporciona el canal en la pata *ZC/TO* es igual a [*periodo del reloj del sistema x prescalar (16/256) x constante de tiempo*].

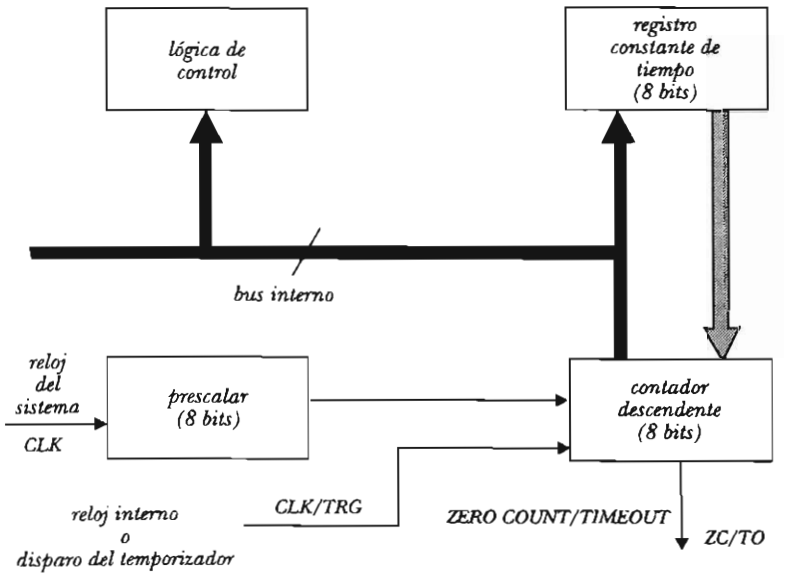
Programación del Z80-CTC

Programar al *Z80-CTC* es enviarle una *palabra de control* que le indique: el *modo de operación* (*contador* o *temporizador*); el valor del *prescalar* (sólo en modo temporizador); si solicitará una interrupción

cuando la *constante de tiempo* de cualquier canal llegue a 0 (teniendo prioridad la del *canal 0*); el tipo de transición activa de la señal suministrada por el pin *CLK/TRG*; si la operación del canal fuera disparada con un pulso en el pin *CLK/TRG* o automáticamente después de cargar la *constante de tiempo*; también permite detener la operación del canal en cualquier momento e indicarle que después de recibir la *palabra de control*, se le enviará al canal una *constante de tiempo*.

La *constante de tiempo* se le envía al canal inmediatamente después de la *palabra de control*, siempre y cuando se le haya indicado anteriormente. La *constante de tiempo* se diferencia de una *palabra de control* porque el bit menos significativo es '0' lógico.

Figura 5.3 Diagrama de bloques de un temporizador típico



Finalmente, podemos decir que resta al programador realizar los cálculos de las *constantes de tiempo* apropiadas para obtener los periodos de tiempo requeridos o para contar el número de eventos externos necesarios.

Trabajo a desarrollar

Adicionar a la circuitería de la práctica anterior un *Z80-CTC (Counter/timer circuit)*, de manera que se lleve a cabo lo siguiente:

1. El *programa principal* transmitirá el código *ASCII* de una *'U' (55H)* en forma serie (a través de un *USART 8251A*). Para transmitir el código *ASCII* de la siguiente *'U'*, el sistema deberá esperar un tiempo no definido, recibir el código *ASCII* de la *'U'* enviada antes, y así sucesivamente se llevarán a cabo estas acciones de manera continua. Deberá utilizarse el método de exploración continua (*polling*) de las banderas *TxRDY* y *RxRDY* del *8251A* y no usar interrupciones.
2. Cada vez que se active un sensor se deberá de interrumpir la *CPU Z80* y la *rutina de atención* a dicha *interrupción* mostrará en el *display 5* cada número de sensor activado durante un período de tiempo (la duración de cada número exhibido en el *display 5* será de *un segundo*).
3. Si se activa más de un sensor, los números que se mostrarán en el display 5 se desplegarán de manera alternada y continua hasta que suceda otro evento; es decir, hasta que se active(n) otro(s) sensor(es), en cuyo caso los números que aparecen en el display 5 podrán cambiar.
4. Esta práctica se caracteriza porque ahora el reloj de tiempo real se mantiene actualizado mediante un *Z80-CTC*, el cual interrumpirá la *CPU Z80* cada segundo.

De alguna manera, las acciones anteriores ya fueron realizadas en la práctica anterior, con excepción de la marcada en el punto 4.

Notas:

* Se probará que, en efecto, el *Z80-CTC* se encuentre interrumpiendo la *CPU Z80* cada segundo, colocando la punta del osciloscopio en la entrada INT de la *CPU*.
* La *CPU Z80* se deberá programar y trabajar en *modo 2 de interrupción*, en cuyo caso el *Z80-CTC* le deberá enviar un *vector de interrupción* cada vez que se le reconozca su solicitud de *interrupción*.

* Es recomendable utilizar más de un *canal del Z80-CTC* para poder interrumpir a la *CPU Z80* cada segundo.
* En esta práctica pueden colocar un 'puente' ente las líneas *TxD* y *RxD* del *USART 8251A* para 'simular' la recepción de datos serie. Es decir, cada vez que el 'puente' sea removido, la *CPU* esperará de manera continua que el código *ASCII* enviado anteriormente le sea devuelto y dejará por lo tanto de transmitir datos por la línea *TxD*.

Figura 5.4 Diagrama de bloques del sistema completo

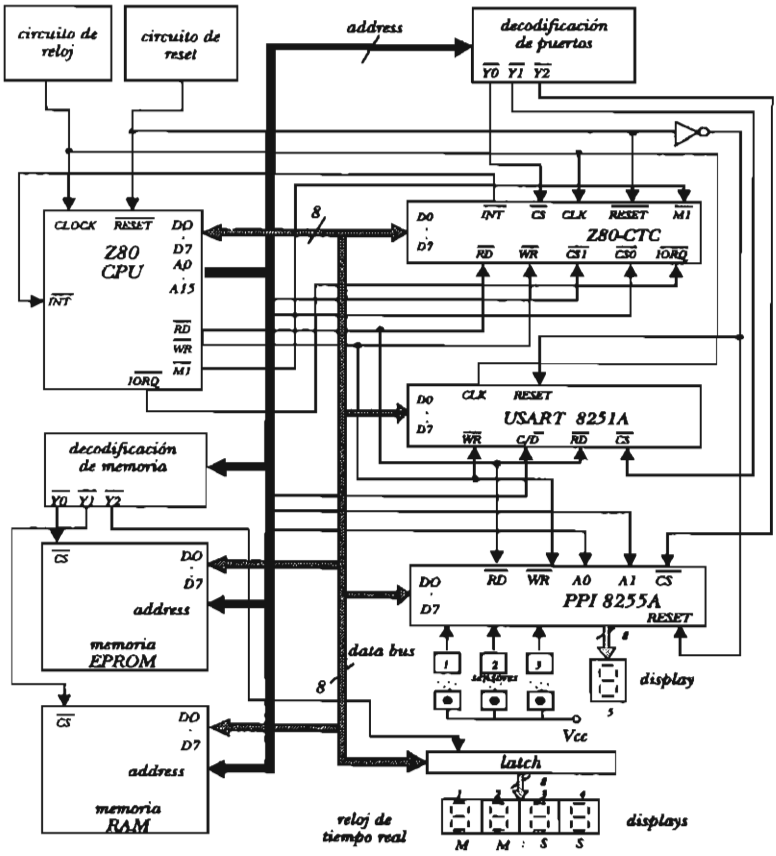


Figura 5.5 El USART 8251

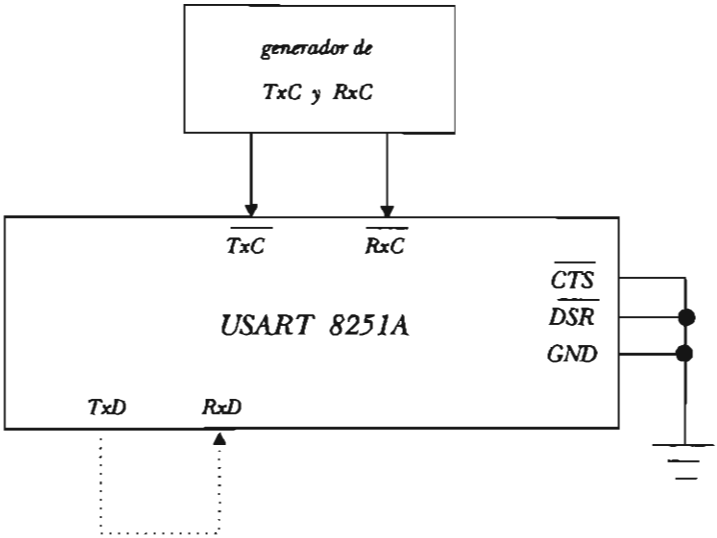
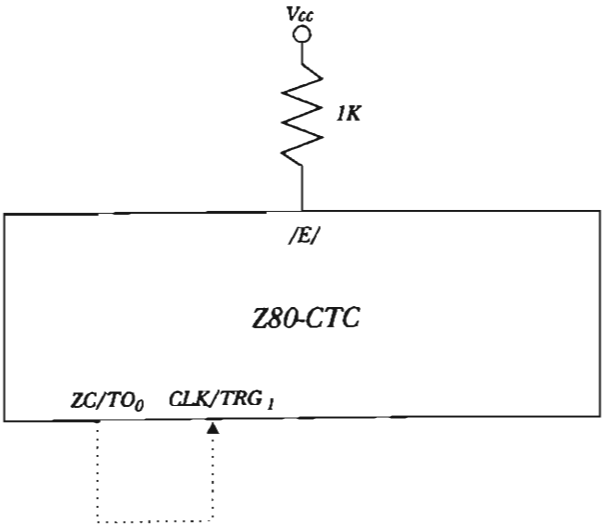


Figura 5.6 El Z80-CTC como temporizador utilizando dos canales en cascada



Práctica 6

Contadores de eventos (el Z80-CTC parte II)

Objetivos

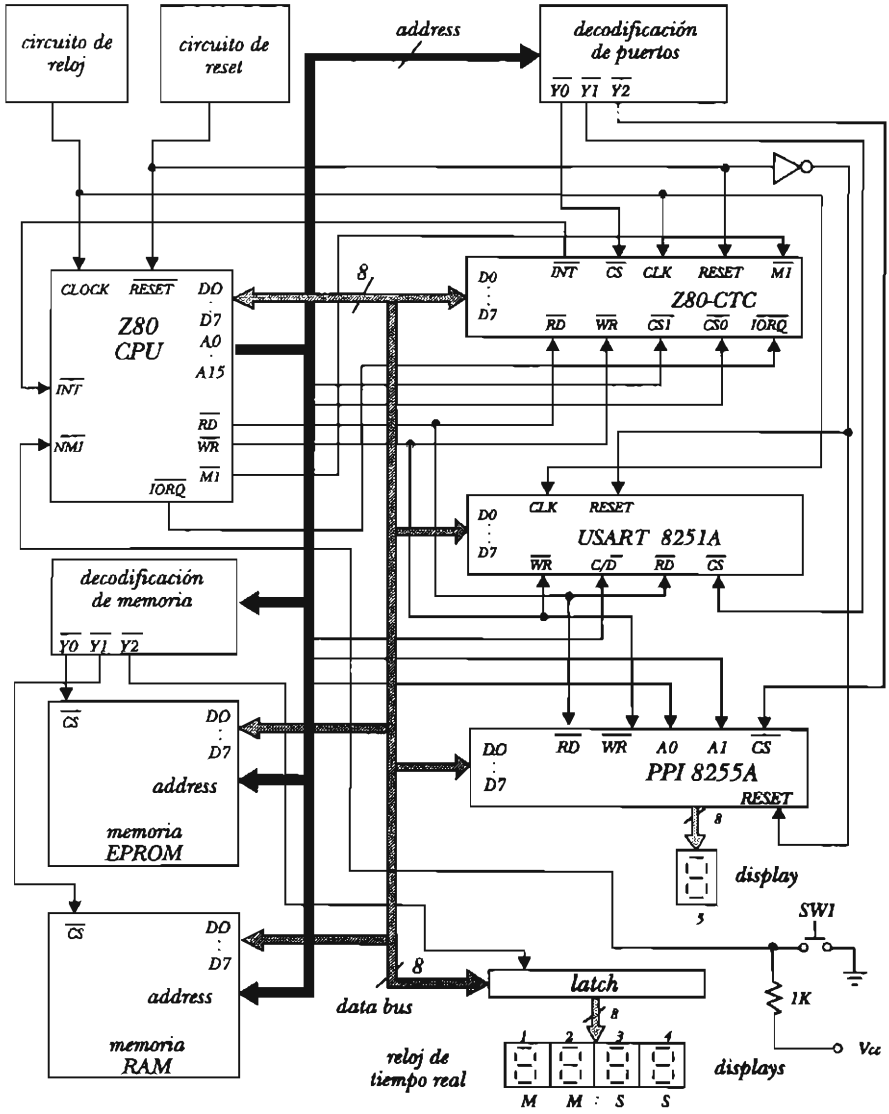
- Adquirir experiencia en el manejo de circuitos contadores (*counter's*) de eventos.
- Manejar de manera correcta la Interrupción no mascarable (NMI: *Non maskable interrupt*) de la Z80-CPU.

Trabajo a desarrollar

Adicionar al sistema digital de la práctica anterior un *switch (SW1)* del tipo *push-button*, de forma tal que se realicen las siguientes operaciones:

1. El *programa principal* se encontrará transmitiendo de manera continua el código *ASCII* de la *letra 'U'*, tal y como se especifica en el punto 1 de la práctica anterior.
2. El reloj de tiempo real (solicitado en la práctica anterior) deberá, de igual forma, ser implementado utilizando el *Z80-CTC* (punto 4 de la práctica número 5).
3. Por otro lado, cada vez que se presione el *switch SW1*, se solicitará una *interrupción no mascarable (NMI)* a la *CPU Z80*, donde la *rutina de atención* a dicha interrupción deberá de leer el valor de un contado y mostrar su estado durante un tiempo $t = 5$

Figura 6.1 Diagrama de bloques del sistema completo



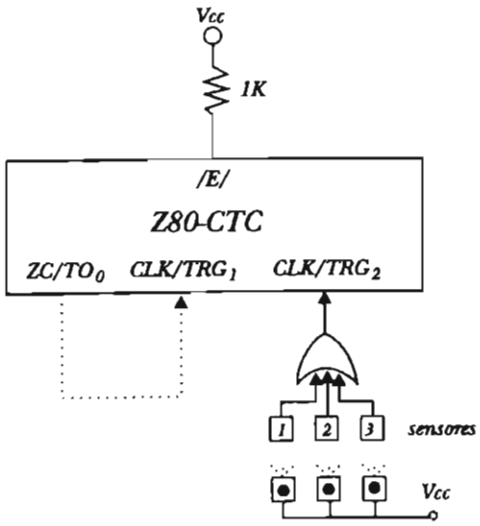
segundos en el *display 5*. Este contador se incrementará cuando uno de los tres sensores sea activado.

El contador al que se refiere el párrafo anterior debe ser implementado y obtenido usando un canal del *Z80-CTC* programado en *modo counter*. Esto explica que cuando un sensor sea activado, también se active la entrada *CLK/TGR* del canal utilizado para actualizar el contador. Es decir, se llevará una cuenta global de cuántas veces se han activado los sensores, la cuenta será leída de un canal del *Z80-CTC* por la *CPU* y mostrada en el *display 5* durante *cinco segundos*.

Notas:

- El valor máximo que podrá tomar el contador será de *255*.
- Cuando se active más de un sensor al mismo tiempo, el contador se incrementará en una unidad.
- El intervalo de tiempo de *t = 5 segundos* se podrá obtener con el reloj de tiempo real implementado en el punto 2.

Figura 6.2 El Z80-CTC como contador de eventos



Práctica 7

Conversión digital-analógica

Objetivos

- Conocer las bases de la conversión digital-analógica.
- Implementar algoritmos para generar una señal analógica usando un convertidor digital-analógico (DAC) en circuito integrado.
- Adquirir experiencia en el manejo de un DAC mediante una CPU Z80.

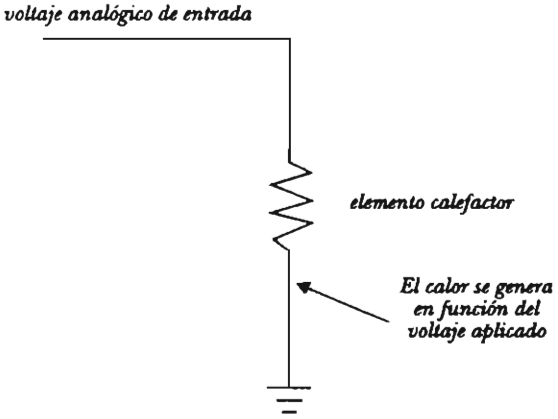
Introducción teórica

Conversión digital-analógica

Cuando se conecta un *microprocesador* a dispositivos externos, algunas veces éstos requieren una señal analógica de entrada por ser controlados. El *microprocesador* es un sistema completamente digital, el mundo real es analógico, por lo que en muchas aplicaciones de control se generan señales analógicas a partir de un sistema digital. A tal proceso se le llama *conversión digital-analógica*, o simplemente *DAC*.

Un ejemplo de estos dispositivos que requieren un voltaje analógico de entrada para controlarlos pueden ser los motores de DC, cuya velocidad de rotación aumenta conforme se incrementa el potencial que se les aplica.

Figura 7.1 Ejemplo de un elemento calefactor controlado por una señal analógica



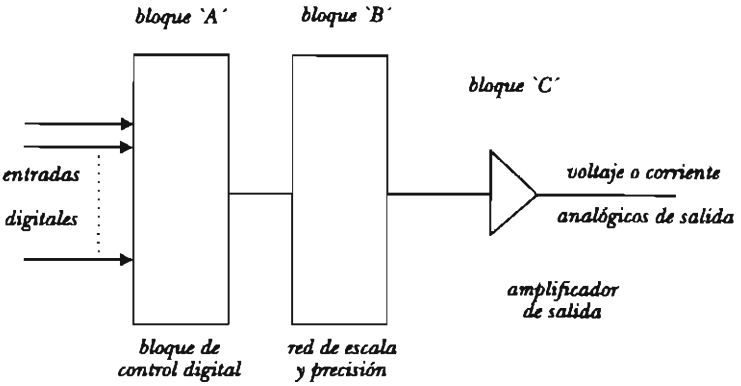
En la figura siguiente se muestra un diagrama de bloques de un *convertidor digital-analógico* típico. El bloque 'A' es la interfaz con la lógica digital que controlará al convertidor. El bloque 'B' es la llamada red de escala y precisión del convertidor, la cual determinará la cantidad de corriente o voltaje de referencia (bloque 'C') que aparecerá en la salida, dependiendo de la entrada digital.

Para ilustrar esto, supongamos que la referencia del convertidor es una fuente de voltaje. El voltaje de referencia es igual a + *10 V.* Usando la red de escala (bloque 'B') se puede forzar a que la salida sea una porción del voltaje de referencia.

Generalmente, los convertidores digital-analógicos presentan un voltaje de salida llamado *voltaje a escala completa*, el cual se obtiene cuando todas las líneas de entrada digitales son '1' lógico. El valor de este voltaje es igual al valor del voltaje de referencia, aunque cada convertidor tiene sus propias especificaciones.

Así también, supongamos que cuando todas las entradas digitales son '0' lógico, el voltaje de salida del convertidor es *0 V.* Un *convertidor digital-analógico* con estas características de voltaje de

Figura 7.2 Diagrama de bloques de un convertidor digital-analógico típico



salida se llama unipolar, debido a que el voltaje de salida crece en una sola dirección, en este caso de *0* a *+ 10V.*

La siguiente información que necesitamos conocer del convertidor es el cambio mínimo de voltaje que se puede obtener en la salida, el cual dependerá de cuántas líneas de entrada se usan para controlar la red de escala.

Si se usan diez líneas de entrada, o diez bits, para controlar la red de escala, el cambio mínimo que se puede lograr en el voltaje de salida se determina dividiendo el voltaje diferencial máximo en la salida entre el número máximo de estados digitales únicos en la entrada. Para este ejemplo, los valores serán los siguientes:

VOUT min = 0.0 V *VOUT max = + 10 V*

Salida máxima diferencial = 10-0 = 10 v

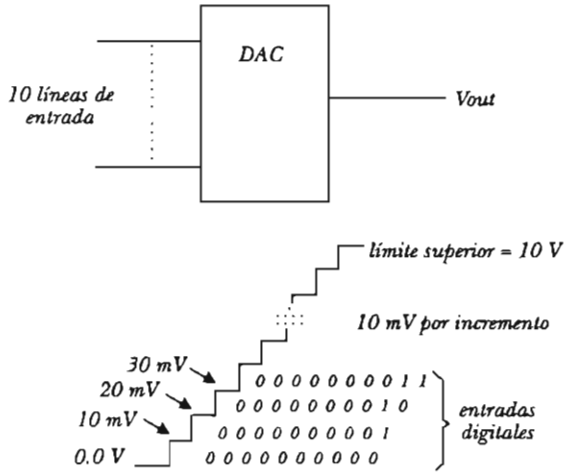
Número máximo de estados digitales únicos $= 2^{10} = 1024$

(Combinaciones posibles con las líneas de entrada)

Sustituyendo valores, se tiene que:

(10-0) / 1024 = 10 mV, cambio mínimo en el voltaje de salida
máximo número
cambio máximo de
de v. estados

Figura 7.3 Incrementos mínimos de voltaje con un DAC



Así entonces, el voltaje de salida cambiará *10 mV* para cada cambio en el bit, menos significativo de las entradas digitales.

Corriente de salida de un DAC

Algunos *convertidores digital-analógicos* entregan corriente en lugar de voltaje. La corriente de salida también está en función de las entradas digitales. Estos *DACs* usan una corriente de referencia en lugar de un voltaje de referencia.

Para ilustrar esto, podemos tomar como ejemplo real al *DAC0808* de *National*, el cual es exactamente igual *DAC M1508 0 M1408* de *Motorola*.

Este *DAC* entrega una corriente de salida la cual es una porción de la corriente de referencia. Esta corriente de referencia se obtiene por medio de un voltaje de referencia y un resistor. Este *DAC* tiene también ocho entradas digitales, de la A1 a la A8, siendo A1 la más significativa.

Para fijar la corriente de referencia del *DAC* a *1 miliamper*, se puede usar una referencia de voltaje de *+ 15 Volts* y una resistencia de *1.5 Kohms*. Esta resistencia puede ser un potenciómetro para poder ajustar el valor exacto de la corriente. Si se desea que el *DAC* proporcione un voltaje de salida se le puede conectar a la salida un *convertidor corriente-voltaje*, el cual se puede obtener fácilmente con un amplificador operacional.

La ecuación que describe la corriente de salida es:

$$I_{SALIDA} = I_{REF} (1/28 \times \text{sumatoria de las entradas digitales A1 - A8})$$

A1-A8 son las entradas digitales del *DAD*. Por ejemplo, supongamos que: *A1=0, A2=1, A3=1, A4=0, A5=0, A6=1, A7=0, A8=0*.

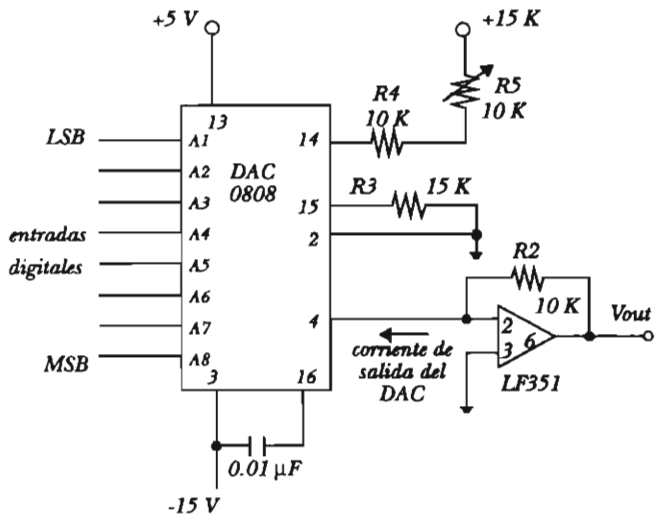
La sumatoria de los pesos o valores de cada uno de los bits es igual a 64 + 32 + 4 = 100, y por lo tanto la corriente de salida será:

$$I_{REF}[(1/256) \times 100] = 1\ mA\ [100/256] = 100\ mA/256 = 0.39\ mA$$

Para asegurar que el valor real de la corriente de referencia sea igual al valor teórico con el cual se hacen todos los cálculos, se puede seguir el siguiente procedimiento:

1. Colocar todas las entradas digitales del *DAC* a '1' lógico.
2. Ajustar el potenciómetro, usado en el voltaje de referencia, hasta que el valor del voltaje a la salida del *convertidor corriente-voltaje* usado sea de *+ 10 V*, con lo cual se asegura que la corriente de salida a través del convertidor corriente-voltaje sea de *1 miliamper*.

Figura 7.4 Conversión corriente-voltaje en un DAC



Resolución de un DAC

La resolución de un *DAC* depende básicamente de la cantidad de líneas digitales que tenga, la cual marca la cantidad mínima de voltaje o corriente de salida que entrega el *DAC.*

Finalmente, debemos mencionar que existe una gran variedad de *convertidores digital-analógicos* de diferentes fabricantes; variedad que se origina porque cada *DAC* tiene sus propias características que indican la rapidez y resolución del mismo.

Trabajo a desarrollar

Adicionar a la circuitería de la práctica anterior un *convertidor digital-analógico* (*DAC*) y realizar un programa que trabaje de la siguiente manera:

1. El *programa principal*, al igual que en la práctica anterior, se encontrará transmitiendo de manera continua el código *ASCII* de la *'U'* en forma serie, pero sin esperar a recibir también en forma serie el *ASCII* de esa *'U'* para poder enviar la siguiente.
2. De manera similar a la práctica anterior, cuando se presione el *push-button SW1*, se mostrará en el *display 5* el valor del contador que indica cuántas veces se han activado los sensores.
3. Además, el *programa principal* deberá encargarse de generar (*por medio del DAC*) un tipo de señal analógica que dependerá del valor presente en el contador. Si el contador es mayor o igual a 5 y menor a 10, el *DAC* deberá entregar una señal *cuadrada*; si el valor del contador es mayor o igual a 10 y menor a 15, el *DAC* generará una señal *triangular* y cuando el valor del contador sea mayor o igual a 15, la señal generada deberá ser *senoidal*.

En resumen, se tendrá:

- *Un programa principal* en el cual se estará transmitiendo de manera continua el Código *ASCII* de la *'U'* (por medio del *USART 8251A*) y monitoreando el valor del contador para generar, dependiendo de su valor, una señal analógica.
- Dos *fuentes de interrupción*:
  - La que solicita el *Z80-CTC* para mantener actualizado el reloj de tiempo real.
  - La *no mascarable (NMI)* que solicita el *push-button*, mediante la cual se muestra durante *cinco segundos* el valor actual del contador.

Para mejorar la forma de la señal analógica es necesario refrescar continuamente la información hacia el *DAC* mientras *TxRDY = '0' lógico*.

Desde otro punto de vista, cada vez que se genere un 'punto' de la señal analógica o se envíe un dato al *DAC*, se deberá de monitorear

el estado de la bandera *TxRDY*. Si dicha bandera se encuentra activada, se deberá de enviar el *ASCII* de la *'U'* al *8251A* (en caso contrario, se deberá de enviar el siguiente 'punto' de la señal analógica al *DAC*).

Notas:

* Los valores de la amplitud y frecuencia de las señales analógicas generadas son a escoger.
* Se puede utilizar cualquier tipo de *DAC*.
* No deben existir discontinuidades en las señales analógicas generadas, lo cual se puede lograr conectando el *DAC* a un puerto de salida del *PPI 8255A*.
* Procurar que las señales analógicas generadas (sobre todo la senoidal) sean lo más fieles posibles.
* Generalmente un *DAC* entrega una corriente de salida, por lo que es necesario adicionarle un *convertidor corriente-voltaje*.

Figura 7.5 Diagrama de flujo del programa principal

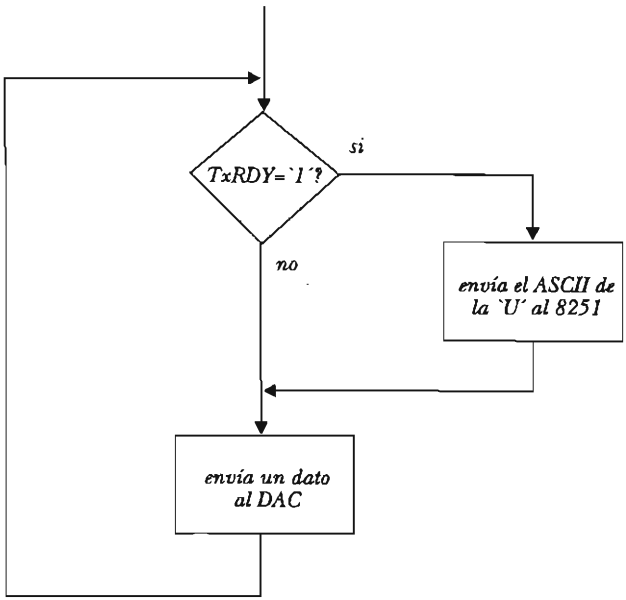


Figura 7.6 El convertidor digital-analógico (DAC)

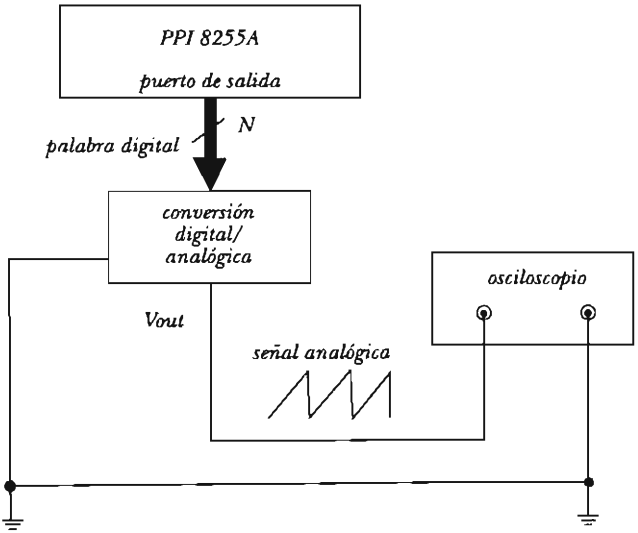
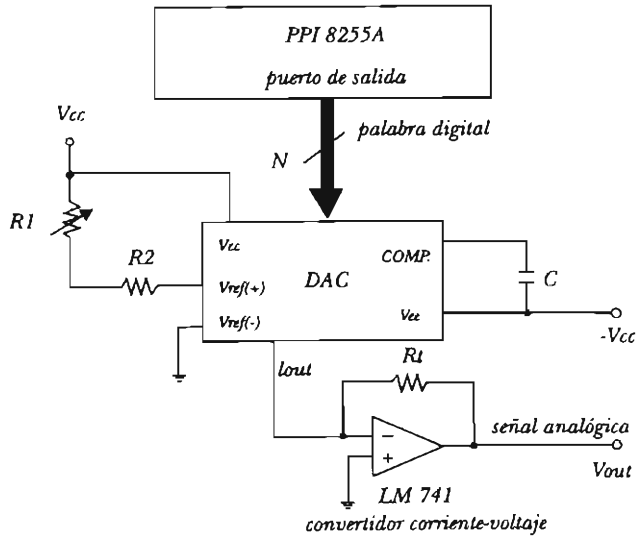


Figura 7.7 El convertidor corriente-voltaje



Práctica 8

Conversión analógica-digital

Objetivos

- Conocer las bases de la conversión analógica-digital.
- Adquirir experiencia en el manejo de un convertidor analógico-digital (ADC) mediante una CPU Z80.

Introducción teórica

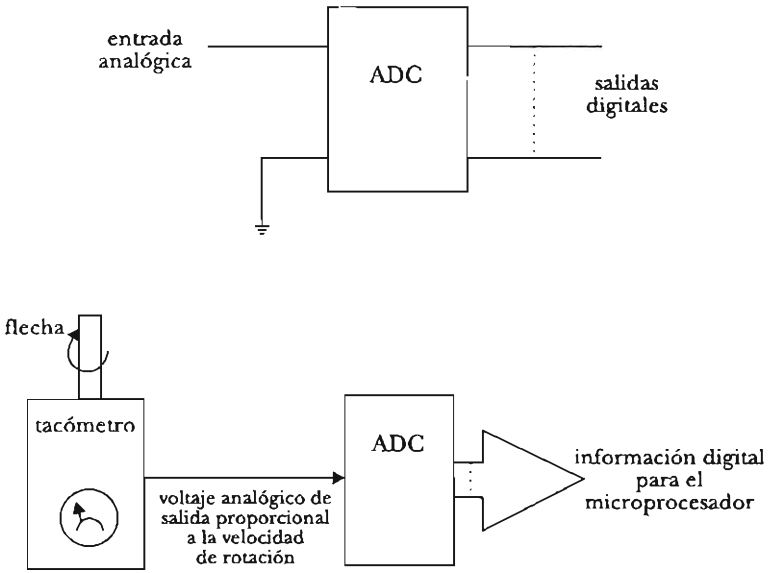
Conversión analógica-digital

En la práctica anterior vimos que en la *conversión digital-analógica* una entrada digital puede determinar eléctricamente el valor de una señal analógica de salida. La *conversión analógica-digital (ADC)* es exactamente lo opuesto, donde la entrada es una señal analógica y la salida es una representación digital de esa señal.

Un *convertidor analógico digital* es útil cuando conectamos *microcomputadoras y microprocesadores* a dispositivos externos que entregan una señal analógica. Usando un *ADC* una computadora puede leer el valor digital equivalente de esa señal analógica de salida de un dispositivo externo y después de eso, la computadora puede ejecutar acciones basadas en el valor digital de la señal.

En la figura 8.1 se muestra cómo un *ADC* puede conectarse a un

Figura 8.1 Ejemplo de una aplicación de un ADC



microprocesador, siendo el dispositivo externo, en este ejemplo, un tacómetro.

El tacómetro entrega un voltaje analógico de salida cuyo valor es proporcional a la velocidad de rotación. Para saber la velocidad a la cual se encuentra trabajando el motor, la computadora debe de leer las revoluciones por minuto del motor. El voltaje analógico de salida del tacómetro es una entrada al *convertidor analógico-digital*, el cual entrega un valor digital al *microprocesador*. Esas salidas digitales del *ADC* componen un número binario que es equivalente a la velocidad del motor.

Esto es solamente un ejemplo de cómo un voltaje analógico puede ser leído por un *microprocesador* vía un *convertidor analógico-digital*.

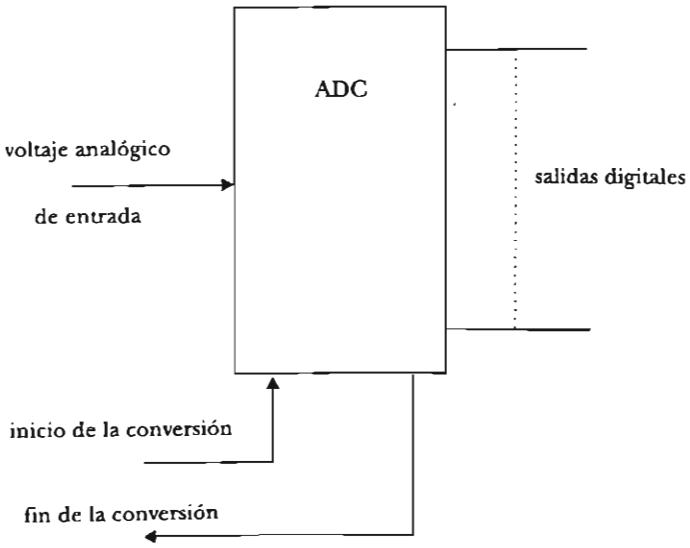
Bases de la conversión analógica-digital

En la figura 8.2 se muestran algunas de las entradas y salidas con las que cuenta un *ADC*. La entrada principal es el voltaje analógico y es la señal que se debe convertir a un valor digital. Las salidas son las que proporcionan el equivalente del voltaje analógico de entrada.

Existen muchos *ADCs* comercialmente disponibles, los que usualmente son mucho más caros que los *DACs*, ya que un *convertidor analógico-digital* usa internamente un *convertidor digital-analógico*.

En la figura 8.2, se pueden ver también dos señales adicionales que son: la *entrada de inicio de la conversión (SCI: Start conversion input)* y la *salida de fin de la conversión (EOC: End of conversion output)*, ya que a diferencia del *DAC*, el *convertidor analógico-digital* requiere una señal de entrada externa que le indique el momento en el que el voltaje analógico se encuentre listo en la entrada para la conversión. Por otra parte, debido a que la operación de la conversión toma una cantidad de tiempo finita, es decir, que físicamente el *microprocesador*

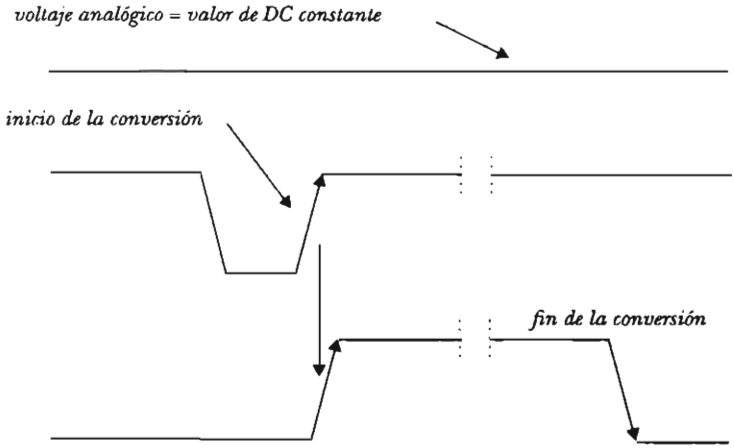
Figura 8.2 Diagrama conceptual de un ADC



debe de esperar que el voltaje analógico sea convertido a valores digitales antes de leer las salidas del *ADC*, el *convertidor analógico-digital* tiene una salida que indica al hardware externo que la conversión se ha completado.

En la figura 8.3 se muestra un diagrama general de la activación de las señales de *inicio y fin de la conversión*. Cada *ADC* puede tener esas señales etiquetadas de manera diferente, sin embargo, realizan la misma función.

Figura 8.3 Diagrama de tiempos de las señales de inicio y fin de la conversión



Las salidas digitales de un *ADC* pueden estar representadas en diferentes códigos, por lo que no hay una técnica estándar para el formato de salida. Las salidas pueden ser generadas con lógica positiva o con lógica negativa. Los códigos típicos usados en las salidas digitales son *BCD, binario, complemento a 1 y complemento a 2*. La hoja de especificaciones del fabricante del *ADC* usado indica el tipo del formato de las salidas. De igual forma, el tiempo de retardo de la conversión varía de un *ADC* a otro. Generalmente, una conversión más rápida hace más caro al convertidor y as

también, entre más bits tenga a la salida el *ADC*, tendrá más resolución y será más caro.

Construcción de un ADC a partir de un DAC

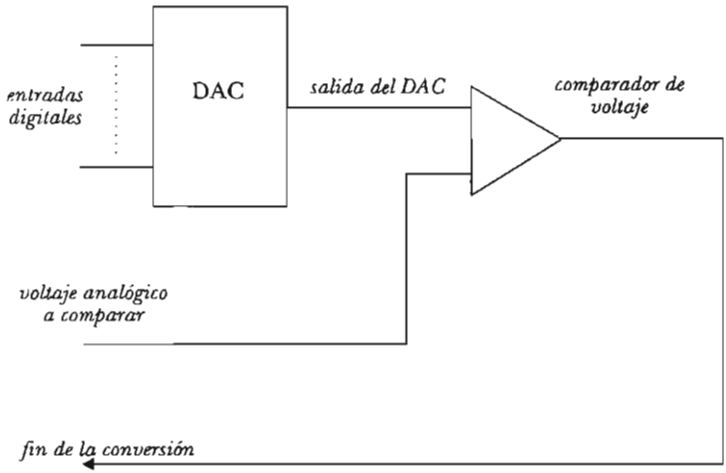
En el siguiente ejemplo se podrá entender cuál es la base con la que trabaja internamente un *ADC*, en el que la salida de un *DAC* se conecta a la entrada de un comparador de voltaje. La otra entrada del comparador es el voltaje analógico que se desea convertir. El voltaje de salida del *DAC* se compara entonces con el voltaje analógico de entrada. Cuando esos dos voltajes son iguales o ligeramente diferentes, la salida del comparador indicará que el valor de las entradas digitales es el equivalente digital del voltaje analógico de entrada.

El *microprocesador* comienza a incrementar la palabra digital de entrada del *DAC* desde 0. Después de cada incremento digital que es enviado al *DAC*, el *microprocesador* checa el estado lógico de la salida del comparador de voltaje. Si dicha salida es '1' lógico, entonces el *microprocesador* incrementa la palabra digital de entrada del *DAC*. El *microprocesador* continuará incrementando los valores hasta que la salida del comparador de voltaje conmute de '1' a '0' lógico.

Antes de usar un *ADC* se debe de investigar cuál es el valor máximo del voltaje analógico de entrada que acepta y cuál es la cantidad de corriente máxima que maneja en sus salidas, lo cual se encuentra en las hojas de especificaciones del fabricante.

Finalmente, al igual que los *DACs*, los *ADCs* pueden ser unipolares (el voltaje analógico de entrada puede tomar sólo valores positivos o sólo valores negativos respecto a tierra), o *bipolares* (el voltaje analógico de entrada puede tomar valores positivos y negativos respecto a tierra), por lo que queda a elección del usuario trabajar con el *ADC* que sea más adecuado a sus necesidades y que cumpla con la rapidez y resolución requeridas.

Figura 8.4 Diagrama de bloque de un ADC construido con base en un DAC



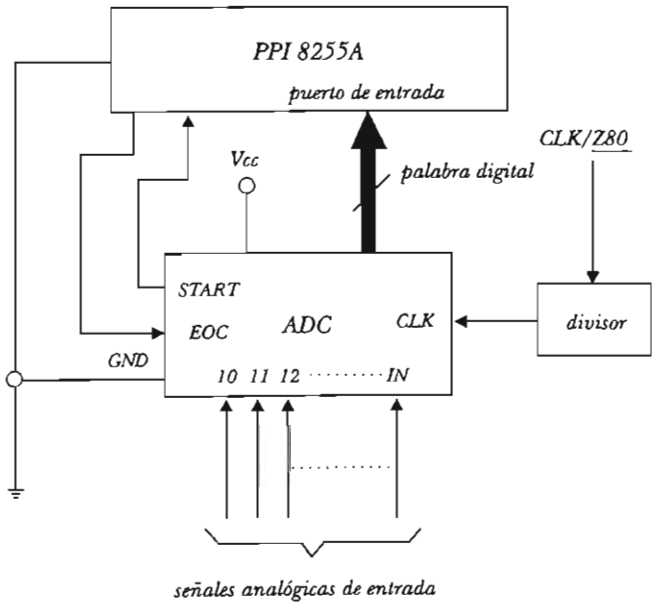
Trabajo a desarrollar

Adicionar al sistema digital de la práctica anterior un *convertidor analógico-digital (ADC)* que tenga al menos tres canales de entrada, de manera que el programa del sistema realice las siguientes acciones:

1. Cada vez que se presione el *push-button SW1*, se deberá solicitar una interrupción no mascarable ($\overline{\text{NMI}}$) a la CPU, para mostrar el valor del contador (número de veces que se han activado los sensores) en el *display 5*.
2. Se deberá mantener actualizado el reloj de tiempo real mediante la interrupción solicitada por el *Z80-CTC*.
3. El programa principal se encontrará transmitiendo (por medio del *USART 8251A*) el código *ASCII* de la letra *'U'* de manera continua, tal y como se realiza en la práctica anterior.
4. Se utilizará un *ADC* que tenga al menos tres canales de entrada porque dependiendo del número del último sensor activado, será

el número del canal del *ADC* a muestrear. La señal analógica de entrada que se esté muestreando deberá de regenerarse utilizando el *DAC* de la práctica anterior. Por ejemplo, si el último sensor que se activó fue el número 2, se deberá de muestrear el canal número 2 y obtener la señal analógica de entrada de ese canal, por medio del *DAC*, es como si se colocara un 'puente' entre la palabra digital entregada por el *ADC* y la palabra digital de entrada al *DAC*. Es decir, el *programa principal* tomará la salida del *ADC* y se la entregará al *DAC*.

Figura 8.5 El convertidor analógico-digital (ADC)



Aquí también como en la práctica anterior, si la bandera *TxRDY* del *8251A* no se encuentra activada, el *programa principal* no deberá de esperar ociosamente a que se active dicha bandera, sino que por el

contrario, deberá muestrear el canal en turno del *ADC* y enviar el valor digital obtenido al *DAC* para, a continuación, volver a probar el valor de *TxRDY*.

Notas:

* Se puede utilizar cualquier tipo de *convertidor analógico-digital*, pero se recomienda usar el *ADC0809* por ser uno de los más económicos y que se puede conseguir fácilmente.

* Inicialmente el *programa principal* no deberá de estar muestreando canal alguno del *ADC*, ni enviar datos al *DAC*, hasta que se active algún sensor.

Bibliografía

Barden, William. *The Z80 Microcomputer Handbook.* Howard W. Sams & Co. 1980.

Coffron, James W. *Z80 Applications.* SYBEX Corporation. 1983.

García Narcia, Octavio F. *Microprocesadores Z80 e Interfaces.* Bioediciones. 1980.

Hayes, John P. *Diseño de Sistemas Digitales.* McGraw-Hill. 1986.

Leventhal, Lance A. *Z80 Assembly Language Programming.* Osborne/McGraw-Hill. 1979.

Mano, M. Morris. *Diseño Digital.* Prentice Hall. 1988.

Rooney, Víctor M. *Microprocessors and Microcomputers.* Macmillan Publishing Company. 1984.

Uruñuela M., José María. *Microprocesadores, Programación e Interconexión.* McGraw-Hill. 1989.

Manuales de los fabricantes

Communication Products Handbook. Western Digital Corporation. 1984.

Linear Databook. National Semiconductor Corporation. 1980.

Manual de Zilog. Zilog Inc. 1977.

Memory Components Handbook. Intel Corporation. 1985.

Memory Data. Motorola Inc. 1984.

Microsystem Components Handbook. Intel Corporation. 1985.

TTL Logic Data Book. Texas Instruments Inc. 1988.

Prácticas de laboratorio de sistemas digitales se terminó de imprimir en mayo de 1996 en Amacalli Editores, S.A. de C.V. Av. México-Coyoacán 421, Col. Xoco General Anaya. Tel. 604 7263. La composición se hizo en Gatineau y Nebraska 12/14. La edición consta de 1000 ejemplares.

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

COSEI
COORDINACIÓN DE SERVICIOS DE INFORMACIÓN

Formato de Papeleta de Vencimiento

El usuario se obliga a devolver este libro en la fecha señalada en el sello mas reciente

Código de barras. 2894630

FECHA DE DEVOLUCION

- Ordenar las fechas de vencimiento de manera vértical.

- Cancelar con el sello de "DEVUELTO" la fecha de vencimiento a la entrega del libro

Roberto Sánchez González

Realizó estudios de ingeniería electrónica en la UAM-Azcapotzalco. Ha impartido cursos de circuitos lógicos, sistemas digitales, organización de máquinas digitales, electrónica III, laboratorio de máquinas digitales, laboratorio de electrónica III, en la misma Universidad. Actualmente sus áreas de trabajo son: redes de computadoras, sistemas operativos y aplicaciones de microcontroladores.

Otros títulos en esta colección

Mariem Henaine-Abed,
Planeación y control de la producción

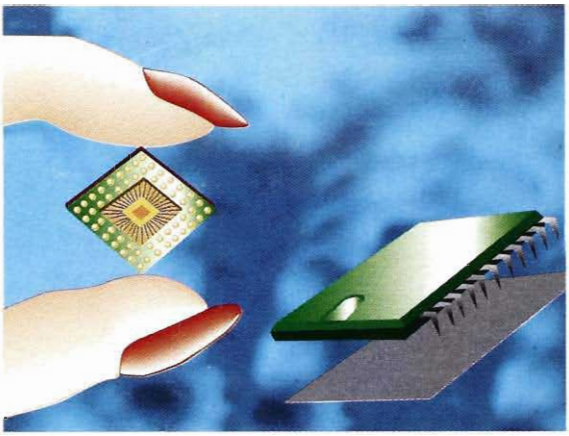
Ana Lilia Laureano Cruces y
Tania Judith Ortiz Ortiz,
Programación orientada a objetos: un enfoque con tipos abstractos de datos y estructuras de datos

Juan José González Márquez,
Introducción al derecho bancario mexicano

Violeta Mugica y José de Jesús Figueroa,
Contaminación ambiental, causas y control

Rafael Quintero,
Electrónica física

Adalberto Cantú Chapa,
Electrónica II. Análisis de diseño con diodos y transistores



La presente obra contiene una serie de prácticas con el microprocesador Z80 e interfaces. Se proporciona al lector los principios básicos en el diseño de un sistema digital cuya base es el microprocesador Z80. Cada práctica tiene una introducción teórica, que explica los conceptos fundamentales y algunas ideas necesarias para la realización de la misma. Se pretende que al terminar de realizar las prácticas se cuente con un sistema digital completo. Los temas que se abarcan en este manual de prácticas son: el microprocesador Z80, lenguaje ensamblador, transmisión paralela, transmisión serie, interrupciones, temporización y conversión digital/analógica y analógica/digital. Por su temática digital y lo actualizado de su contenido, este libro resulta una excelente guía para estudiantes de las carreras de ingeniería electrónica, ingeniería en sistemas e ingeniería en computación.